Proceedings of the 2019 Winter Simulation Conference
N. Mustafee, K.-H.G. Bae, S. Lazarova-Molnar, M. Rabe, C. Szabo, P. Haas, and Y. J. Son, eds.

DOMAIN-SPECIFIC LANGUAGE FOR MODELING AND SIMULATING ACTIONS IN LOGISTICS NETWORKS

Markus Rabe

Department IT in Production and Logistics
TU Dortmund University
Leonhard-Euler-Str. 5
Dortmund, 44227, GERMANY

Dominik Schmitt

Graduate School of Logistics
TU Dortmund University
Leonhard-Euler-Str. 5
Dortmund, 44227, GERMANY

ABSTRACT

Making adjustments to a logistics network to keep it in good condition is a major challenge. Logistics assistance systems are regularly used to support this process. The authors have developed such a logistics assistance system that identifies, evaluates, and proposes promising actions to the decision-maker. A simheuristic approach, utilizing a data-driven discrete event simulation in combination with meta-heuristic algorithms is used for this purpose. A typical feature of such systems, however, is that the possible changes to the logistics network are predefined by the respective logistics assistance system. In order to address this aspect, the authors have developed a novel method that allows for modeling, integration, and simulation of user-generated actions. The method is based on a domain-specific language for the formal description of actions in logistics networks, allowing domain experts to model actions without having in-depth programming knowledge.

1 INTRODUCTION

Wholesale logistics networks (WLNs) have complex structures and a multitude of different processes and interrelationships. Due to changing internal and external requirements, these WLNs must be continuously monitored and controlled, in order not to lose competitiveness. Because of the complexity of WLNs, manual monitoring and controlling is virtually impossible. Therefore, logistics assistance systems (LASs) are regularly used to support the decision-making process. Such systems may identify promising actions, e.g., changing the stock-level of stock keeping units (SKUs) or the adjustment of routes in the network, and propose them to the decision-maker.

The authors developed a LAS for identifying, evaluating, and suggesting promising actions in WLNs. This system utilizes a combination of meta-heuristic algorithms with discrete event simulation (DES), called a simheuristic approach (Juan et al. 2015). Heuristic algorithms explore a set of actions, searching for the most promising ones. The DES serves to evaluate the effects of actions on the performance of the logistics network, which is determined by the service level and the total costs. Typical for such LASs, nonetheless, is that possible actions are predefined by the system. An enterprise-specific adaptation of these actions and their effects to the network is, if at all possible, very time-consuming and associated with high costs. To address this problem, the authors have developed a method for modeling and simulating user-generated actions in WLNs. The main component of this method is a newly developed domain-specific language, which abstracts the technical implementation of these actions to a more logistical level.

The paper is structured as follows: Section 2 introduces the general principles for further understanding this paper. The LAS developed by the authors is presented in Section 3, with a focus on the process of modeling and implementing user-generated actions. Section 4 sets out the requirements for a formal description of action types. On this basis, Section 5 provides a semantic model for representing these action

types. Section 6 presents the newly developed domain-specific language, with a focus on new data types, language constructs, and other functionalities. Section 7 gives a brief conclusion and an outlook.

2 RELATED WORK

2.1 Wholesale Logistics Networks

Looking at the term logistics network and its definition, it can be seen that in the relevant literature there is no clear distinction between supply chains (SCs) and logistics networks (LNWs) (Cordeau et al. 2006; Wang et al. 2016). This view is supported by Larson und Halldorsson (2004). On the other hand, there is the view that a logistics network is exactly that part of a SC that lies between the SC's suppliers and its customers (Rushton et al. 2010), in other words, the logistics network takes the role of the distributor within the SC (Ma und Suo 2006). For the further course of this paper, the authors use the term LNW, as this is the term used in the company involved. In principle, however, the findings of this paper can also be transferred to SCs.

This paper is based on a wholesale logistics network, which is a special type of logistics network. A wholesaler is a company that purchases goods and then resells them, mainly to industrial customers (Seÿffert 1972). WLNs occupy an intermediary position between producer and customer, due to their level of trade and the orientation of their business model (Barth et al. 2015).

2.2 Discrete Event Simulation

For the investigation and analysis of processes and interactions in the context of production and logistics, Discrete Event Simulation (DES) has established itself as the preferred method (Wenzel et al. 2009). In DES, state changes of the simulation model occur due to events that can arise at any time, such as an incoming order or a completed picking process (Gutenschwager et al. 2017). Usually, the simulation is used in conjunction with other methods, e.g., optimization methods. If the optimization is a meta-heuristic and the heuristic is used in conjunction with simulation, it is a so-called simheuristic according to Juan et al. (2015).

2.3 Logistics Assistance Systems

LASs are software systems that are used to support decision-making processes in the context of logistics (Blutner et al. 2009). In the literature, the terms logistics assistance system and decision support system for logistics applications are regularly used synonymously (Kengpol 2008). The authors have chosen the term logistics assistance system because it better highlights the application domain.

A LAS can have different characteristics and applications, e.g., for monitoring or optimizing processes in logistics networks. A typical application field of LASs is the automotive industry, e.g., with a focus on production (Bockholt et al. 2011) or in combination with other methods, such as simulation (Heilala et al. 2010).

2.4 Domain-specific Languages

Programming languages can basically be divided into two classes. Typical for the first class, the general languages (e.g., the Unified Modeling Language, UML) (Object Management Group 2015), is its possible use for a large variety of different problems and the resulting power of the language (van Deursen et al. 2000).

The second class of programming languages are the so-called domain-specific languages (DSLs), e.g., the Structured Query Language (SQL) (Oracle Corporation 2019). Further terms for DSLs are application-oriented (Sammet 1969), special-purpose (Wexelblat 1981), task-specific (Nardi 1993), microscopic or little (Bentley 1986) languages. Yet, the term domain-specific language appears to be the most commonly used name. A DSL is essentially used for solving problems of a specific domain (Voelter 2010 – 2013).

Syntactic gaps between the problems of the corresponding domain and their implementation are bridged by abstraction (Cheng et al. 2015). This allows domain problems to be solved at the domain level (France und Rumpe 2007). The DSL's target group consists of domain experts with little or no programming experience.

According to Fowler und Parsons (2011), a DSL can also be understood as a thin layer over a semantic model. A semantic model consists of data including the model's semantic and all relevant information of the domain. In addition, the behavior, for instance, in the form of functions, is part of the model. In this context, a DSL is used to populate the semantic model. In this paper the developed DSL is used for the parameterization of a semantic model for the representation of actions in WLNs.

3 LOGISTICS ASSISTANCE SYSTEM FOR WHOLESALE LOGISTICS NETWORKS

A LAS for WLNs has been described by the authors in previous publications (Rabe et al. 2018b; Rabe et al. 2019). This section gives a brief overview of the simplified architecture of the LAS (see Figure 1).

Usually, organizations use common standard software to store and access enterprise data, e.g., SAP R/3. The author's LAS extracts the enterprise data and transforms and loads them into the data model of the simulation tool used, which is called SimChain (SimPlan AG 2017). SimChain essentially consists of two components: a set of generic building blocks for simulating logistics networks and an underlying data model stored in a MySQL database. The database contains the information required for the simulation of the logistics network under consideration. Based on this information, a simulation model is created dynamically. This approach makes it possible to manipulate the simulation model at the data level by making changes to the data in the database.

The LAS uses an iterative simheuristic (Juan et al. 2015) approach with a data-driven DES as an evaluation function for the configuration of the logistics network stored in the database. In each iteration, a simulation model is dynamically instantiated based on the data from the database. Subsequently, a simulation experiment is carried out. The simulation results are stored in the database, to which the heuristic unit (HU) has access. The HU selects one or more promising actions from the search space, which consists of all possible actions for the given state of the WLN. For a detailed description regarding the HU's implementation, the reader is referred to Rabe et al. (2018a, 2018b). The selected actions are forwarded to a system component called the execution engine, which transforms the actions into changes to the underlying database in the form of SQL statements. Based on the changed data, a new simulation model is created and executed in order to evaluate the actions' effects on the logistics network's performance, e.g., costs and service level. This overall process is run iteratively until a termination criterion is fulfilled, for instance, a specific number of iterations or stagnation of the WLN's performance. After the simulation-based optimization is terminated, the most promising actions and their effects are suggested to the decision-maker.

3.1 Utilizing a Domain-specific Modeling Language for User-generated Action Types

The presented LAS was extended by a method with which users can model their own action types and integrate them into the system. The essential component of this method is a specially developed language, or more precisely a DSL, which is tailored to the modeling of action types in WLNs. An action type represents a set of similar actions, e.g., an action type may describe the change of stock for any SKU in any site by any value. A corresponding action can be derived by adding specific parameter values, for example a +20 % change in inventory for the SKU with id 5 in the site Dortmund. The DSL for modeling such action types can be accessed via an interface, the so-called action type designer. This user interface is implemented as an integrated development interface (IDE) for the DSL, which provides a modeler with all benefits of common IDEs such as code completion or syntax highlighting. The language constructs of the DSL and any previously defined action types can be used to model new action types. All created action types are stored in the action type directory and, thus, made accessible to the system.

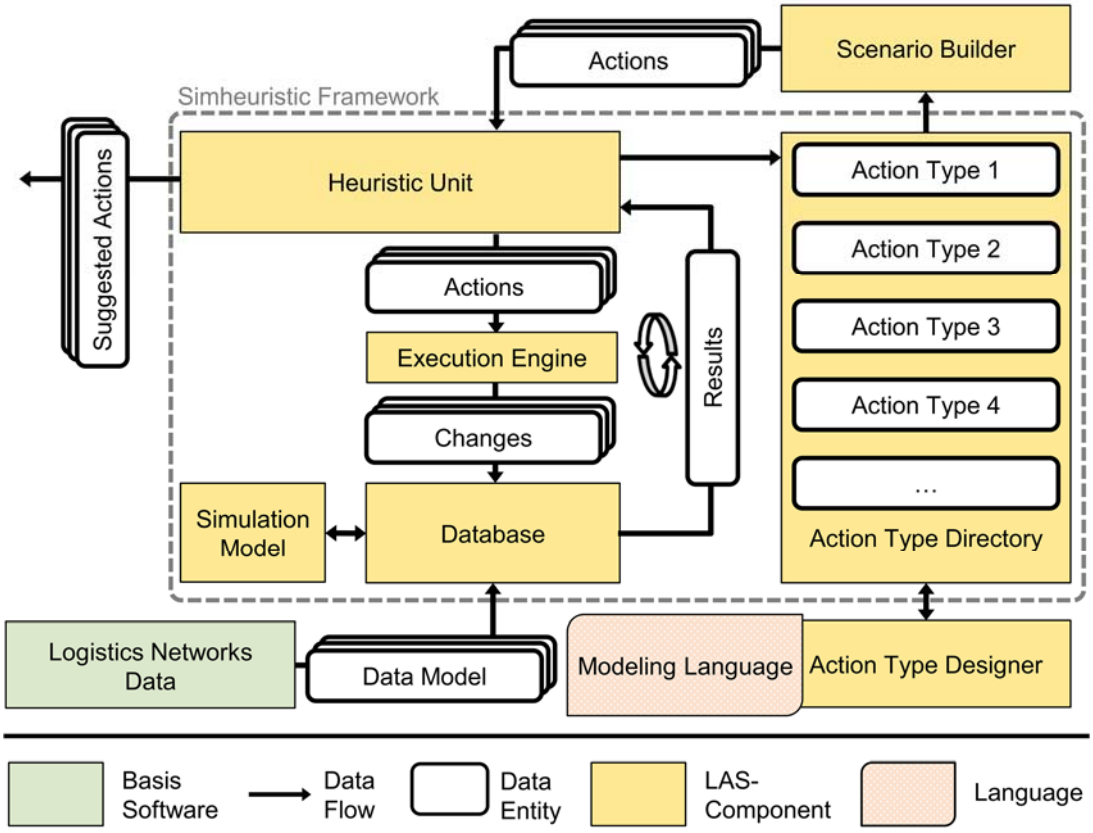

Figure 1: Architecture of the logistics assistance system, based on Rabe et al. (2018a).

3.2 Process of Deriving Actions from Action Types

The simulation scenario can be configured in another user interface, the so-called scenario builder. The scenario builder has access to the action type directory and, therefore, to all actions types stored in it. The user can select any number of action types from the action type directory to be taken into account for the current simulation scenario. After each selected action type has been configured, all corresponding actions are derived and added to the HU's search space (Rabe et al. 2018b).

In addition to the selection and configuration of action types, the user can make further settings. For example, scheduling criteria, optimization targets or other general conditions can be defined. Once the configuration of the scenarios has been completed, the simulation-based optimization can be started.

4 REQUIREMENTS FOR MODELING ACTION TYPES

The effects of an action type on the respective WLN are company-dependent. In addition, action types are subject to constant change and must be continuously adaptable. As a result, a method for modeling action types must ensure that it is also possible to address unknown relationships and effects. In the authors' LAS, action types represent changes to the simulation model by manipulating the underlying database. Accordingly, it must be ensured that all possible changes, in particular adding and removing entities and changing the entities' attribute values, can be mapped to the database using this method.

LASs are usually used by different users, for example by modelers, simulation experts, or decision-makers. The traceability of a formal description of action types and the associated effects on the logistics network are essential criteria for both acceptance and collaboration. As a result, a method for a formal description of action types must be as accessible and comprehensible as possible, e.g., by following a natural language flow.

Action types can have complex effects on the underlying logistics network, for example when an existing site is closed. Transferred to the LAS under consideration, this means that a simple and intuitive method for the formal description of action types must be ensured, e.g., by providing the most powerful constructs possible or by ensuring a natural writing flow during modeling. It should also be noted that it is necessary to include existing action types in the modeling of a new and complex action type in order to ensure that existing action types can be reused. Such a modular approach also reduces the modeling effort and complexity of action types.

An action type, and thereby corresponding actions, can influence several entities of the underlying logistics network, e.g., all SKUs of an assortment. In such a case, an action may change all affected entities in the same way, e.g., by increasing the stocks of all affected SKUs by 20 %. However, in some cases, an individual change may have to be made for each individual entity. For instance, when the sourcing of SKUs is changed, a suitable supplier may have to be found for each SKU and set accordingly as the new supplier of this article. A method for the formal description of action types requires the possibility of being able to map corresponding distinctions and recurring facts as efficiently as possible, e.g., in the form of case distinctions or loops.

5 SEMANTIC MODEL FOR THE REPRESENTATION OF ACTION TYPES

Any action type is built upon the same semantic model. Thus, the semantic model needs to be capable to represent all necessary information for any possible action type in a wholesale logistics network. Differences between the characteristics of different action types are exclusively represented by an individual parameterization of the semantic model's attributes. The initialization of these attributes can be done by different sources, e.g., by a user or by the system and at different stages of the modeling process. The action type's semantic model shown in Table 1 is described in the following.

The attributes of the semantic model are serving various purposes and, therefore, are divided into different categories. The first group of attributes have an informative objective, such as an action type's *id*, a *name*, a *description*, and a *list of ids* of one or more involved modelers, as well as the *owner's id*. An action type's *id* is used as a system-internal identifier and set automatically during the modeling process. The action type's *description* is given by the modeler and utilized to improve the understanding of the action type's implementation and its effects on the underlying wholesale logistics network. The action type's *name*, set by the modeler, and the *modelers' ids*, set by the system, are used by the users of the LAS in order to identify and recognize an action type. In addition, access permissions, e.g., for editing or deleting an action type, can be managed based on the *owner's* and *modelers' ids*.

Action types are representing changes to the simulation model and, therefore, to the underlying wholesale logistics network. For the specification of these changes, functional attributes are used. An action type describes a set of similar actions that differ in the entities they affect. The attribute *input* is used to provide a mechanism for specifying these entities by representing a list of input parameters. In addition, a list of *statements* is used to characterize the type of changes that are applied to these entities. These functional attributes are implemented by the modeler during the modeling process.

Realizing actions in the real wholesale logistics network can entail costs. The total costs of an action type are stored in the attribute *total costs*. Equivalent to the costs, the conversion duration is stored in the attribute *time till effect*. These attributes, representing information related to the execution of an action type, are set by the modeler.

Between the modeling and implementation, an action type passes through several stages, e.g., on hold or under review. The current stage of an action type is stored in the attribute *status* which is updated automatically by the system. The implementation of an action type – the DSL program – is stored in the attribute *program code* automatically. Thus, the code can be reviewed and loaded into an IDE easily, without any compilation or translation. Some action types cannot be executed on their own, but only in connection with other action types. Thus, this information is stored in the attribute *autonomous*, which needs to be defined by a modeler during the modeling process.

Domain-specific information, e.g., the *frequency* or the action type's *impact* on the WLN's performance, can be stored in corresponding attributes. These parameters are set be the modeler. In addition, the sort of changes of the underlying WLN can be divided into two groups: first, structural changes, such as adding or removing entities, and second, configurational changes, such as changing attribute values. This information is stored in the attributes *structural* and *configurational*, accordingly. The sort of changes is determined by the system and set accordingly. Correlations between different action types can also be modeled, e.g., when centralizing an SKU in a site, increasing the stock and safety stock level of that SKU in the given site or increasing the transport frequency of affected routes may be promising candidates for further actions. The correlation between different action types must be specified manually. These domain-specific information can be utilized to guide the search for promising action plans and, therefore, to reduce computational time of the LAS (Rabe et al. 2018a).

For further processing of an action type, any action derived is automatically stored in an attribute called *actions*. These actions depend on the action type's input parameters' values and the current state of the wholesale logistics network. The number of actions across all action types determines the search space of the heuristic unit.

Table 1: The semantic model for the representation of action types in wholesale logistics networks.

Attribute	**Description**	**Source**
Actions	List with all derived actions, depending on the logistics network's state.	System
Autonomous	Specifies whether the action type can be used on its own or only in combination with other action types.	Modeler
Configurational	Domain-specific information that specifies, for an action type, whether the corresponding changes are configurational.	System
Correlation	Domain-specific information that specifies possible correlations and their correlation factor with other action types.	Modeler
Description	Free description of the action type.	Modeler
Frequency	Domain-specific information that specifies the frequency of the implementation of derived actions.	Modeler
Id	Id of the action type.	System
Impact	Domain-specific information that specifies the impact of derived actions to the underlying logistics network's performance.	Modeler
Input	List of input parameters.	Modeler
Modelers	List with ids of the involved modelers.	System
Name	Name of the action type.	Modeler
Owner	Owner of the action type.	System
Program code	DSL script that represents the program code of the action type	System
Statements	List of statements, representing changes to the underlying logistics network.	Modeler
Status	Status of the action type.	System
Structural	Domain-specific information that specifies whether the corresponding changes of the action type are structurally.	System
Time till effect	Required time for a corresponding action to take effect.	Modeler
Total costs	The costs associated with the implementation of derived actions.	Modeler

Any attributes of the semantic model assigned to a modeler are implemented during the modeling process using a domain-specific modeling language. Thus, language constructs for these specific implementations are required.

6 CONCEPT OF A DOMAIN-SPECIFIC LANGUAGE FOR MODELING ACTION TYPES

A domain-specific modeling language can be seen as a thin layer over the semantic model (see Table 1) that is used in order to initialize this model. The DSL script generated during the modeling process is processed by a parser and converted into an abstract syntax tree (AST). Based on the AST, a semantic analysis of the modeled action type is performed and a populated semantic model is generated. The sequence of processing a DSL script for initializing the semantic model is shown in Figure 2.

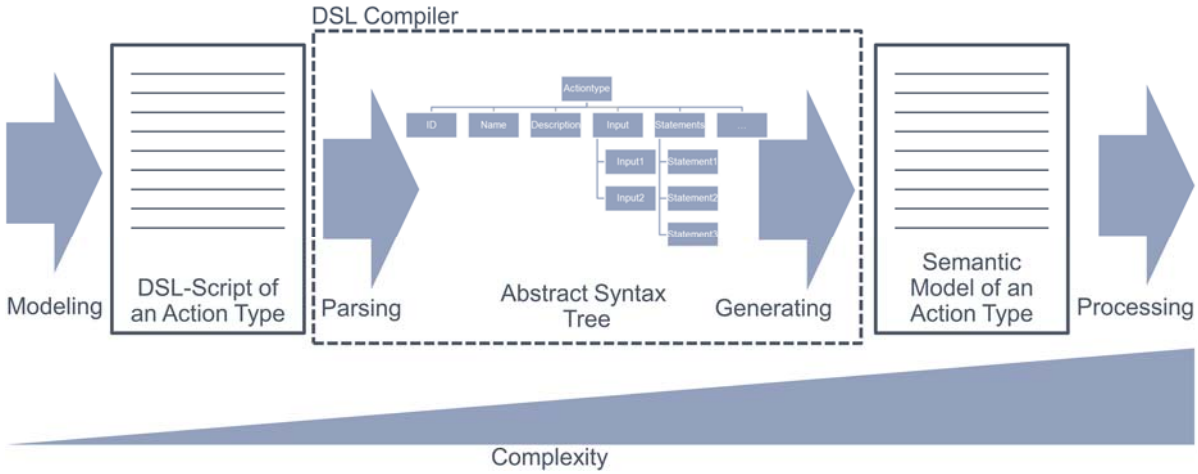

Figure 2: Processing a DSL script and populating the semantic model of action types in wholesale logistics networks.

Domain-specific data types and language constructs in addition to some language constructs from general-purpose languages are used to create a DSL-script that represents the formal description of an action type. The set of language constructs of a DSL should be kept as small as possible, e.g., using only one kind of case distinction and loop. In the following section, the concept of a DSL for modeling action types in WLN is presented. A complete description of the language and its grammar is given in Schmitt (2019).

6.1 General Language Constructs

In addition to some domain-specific language constructs, the DSL also contains some more general constructs. These are essentially similar to corresponding language constructs from general-purpose languages like Python, Java, or C#, so that only a brief introduction is given.

Variables are used to store and retrieve data during the course of an action type. These variables are referenced by a unique identifier and must be assigned to concrete data types during declaration. A value assignment is performed using an equals sign. Otherwise, the variable's value is null.

An if-then-else expression is available to a modeler for making conditional case distinctions. With this language construct, values or ranges of values as well as boundary conditions can be evaluated. The further execution of corresponding actions may depend on the result. If-then-else expressions can be nested to any depth, by adding additional else-branches.

It is possible to use a for-each loop to model a list of recurring statements or declarations. Such a loop passes through all values stored in a specified variable. The loop body is executed exactly once for each element of the variable. Within the loop's body, the current element can be accessed, e.g., for further processing.

The language construct comment can be used to specify additional information in the modeling of action types. A comment is initialized with a hash and can then contain any number of valid characters of the language, until the end of the current line of code. Comments are used to describe the action type and have no influence on its effects on the underlying logistics network.

6.2 New Class of Data Types

Data types are used to store complex related information in a single structure. For modeling action types in WLNs the data types *modifier*, *list*, *filter* and *query* are used, which are presented in the following.

The data type *modifier* is used to represent any changes to attribute values of one or more entities of the wholesale logistics network, e.g., the change of stock level of any SKU in a given site. These manipulations can be defined as absolute values (e.g., 100, Monday, True) or relative values (+10, -7%, 2*(12-myVariable), respectively. Thus, the *modifier* consists of an algebraic sign, a value, and a percentage character. The algebraic sign and the percentage character can be defined optionally, but the percentage character may not be set without a sign.

A *list* can be used to store multiple values or variables, e.g., one or more *modifiers*. These values are enclosed in square brackets and separated from each other by a comma, e.g., [-7, 9-3, myValue, 12]. A *list* can only store values of the same underlying data type, such as Integers, Strings, or Booleans. Nested structures, e.g., a *list* in a *list*, are automatically resolved. Thus, *lists* are always one-dimensional. This ensures the comprehensibility and traceability of data in a data structure.

When manipulating the simulation model by applying an action, the effected entities can be further specified. Therefore, the data type *filter* can be used. For example, when changing the stock of any SKU in a specific site, a *filter* might be used to restrict the affected SKUs to those with a weight of more than 100 kilogram. A *filter* consists of one or more filter criteria. Each filter criteria is defined for a specific entity class' attribute, e.g., for the entity class SKU and its attribute weight. In addition, a condition must be defined for each of these combinations. Such a condition consists of a relational operator and a value, e.g., > 100. Multiple filter criteria are each linked by a logical operator, e.g., by "and" or by "or". For a better structuring of multiple filter criteria, it is possible to use parentheses. Additionally, filter criteria can be negated. In the following, a snippet of a formal description of an exemplary action type is presented, in which a *filter* is declared. The *filter* myWeightFilter has the condition that entities must have at least a value of 100 for the attribute weight of the corresponding entity class SKU.

```
myWeightFilter TYPE FILTER = SKU.weight >= 100
```

The simulation model's entities can be loaded into a data structure called *query*. In such a *query*, a set of entities with the same structure, i.e., entities with the same attributes, can be stored. Therefore, the corresponding entity class must be specified together with the *query*. In addition to the entities, the structure of the associated entity class is stored in the *query* as well. A *query* can also consist exclusively of the structural information of the underlying entity classes, i.e., it can be empty. A *query* is not limited to a single entity class and, thus, can store the structure and entities of multiple related entity classes. To enable this feature, the entity classes are connected by using the logical operator "and". New entities can be added to a *query* that do not necessarily have to originate from the underlying database, but can be freely parameterized by a modeler. Entities that are not related cannot be stored in the same *query*. In the following example, the variable mySKUQuery is declared as a *query* and initialized with all entities of the entity class SKU. The content of the *query* is displayed in Table 2 in parts.

```
mySKUQuery TYPE QUERY = ENTITY SKU
```

Table 2: Partial tabular display of *query* mySKUQuery.

SKU.id	SKU.name	SKU.weight	SKU.height	…
1	SKU1	123	154	…
2	SKU2	26	1169	…
3	SKU3	178	13240	…
4	SKU4	73	948	…
5	SKU5	582	6400	…

6.3 Functions on Data Types

Some of the presented data types have additional functions that can be executed. These functions are used for increasing the flexibility of the corresponding data types and, therefore, of the domain-specific language.

The data type *list* has additional functions for a more specific selection of data from the underlying wholesale logistics network. For example, a modeler can select any number of elements from a specific index or from the beginning or the end of a given *list*. To increase the flexibility, the *list's* elements can be sorted in an ascending or descending order. Furthermore, additional information can be selected from the *list*, such as the count, the sum, the average, the maximum, or the minimum of any corresponding elements. Multiple *lists* can be combined into one single *list*.

Two or more *filters* can be combined into one single *filter* by utilizing logical operators, e.g., by an "and" or by an "or". Additionally, these *filters* can be structured by using parentheses.

The functionalities of a *list* also apply to the data type *query*. Because of the multi-dimensionality of a *query*, an additional tuple, consisting of an entity class and one of its attributes, must be specified to access its information, e.g., the sum of SKU.weight for all entities of a *query*. Furthermore, a *query* can optionally be combined with one or more *filters*. The result of this filtering is a subset of the original *query*, including all entities that fulfill the criteria of that *filter*. The result can be stored in another *query*. In the following example, the *query* mySKUQuery is combined with the *filter* myWeightFilter, both from the previous examples. The result is saved in the new *query* myFilteredSKUQuery and is available for further modeling. In Table 3, the *query's* content is displayed in parts.

```
mySKUQuery TYPE QUERY = ENTITY SKU
myWeightFilter TYPE FILTER = SKU.weight >= 100
myFilteredSKUQuery TYPE QUERY = mySKUQuery
  WITH FILTER
  myWeightFilter
```

Table 3: Partial tabular display of the *query* myFilteredSKUQuery after combining the *query* mySKUQuery with the *filter* myWeightFilter.

SKU.id	SKU.name	SKU.weight	SKU.height	…
3	SKU3	178	13240	…
5	SKU5	582	6400	…

6.4 Language Constructs for Manipulating the Simulation Model

For manipulating the entities' attribute values of the underlying wholesale logistics network, *change statements* are used. A change statement is executed on a single entity class, altering one or more attribute values for all corresponding entities. For this purpose, one or more attributes of the entity class must be defined, which then will be changed. A *modifier* is used to specify the exact changes. A subset of the affected entities can be selected by using a *filter*. To change attribute values, the data type of the respective attribute, e.g., Integer, String, Float, and its value range must be taken into account.

The *add statement* is used to add new entities to the simulation model, e.g., a new SKU to a site. Therefore, an entity class must be defined to which the new entities are added. A specific value can be assigned to each attribute of the entity class. Such an assignment can be made directly by providing a specific value or by a reference to an existing *list*. If a *list* contains more than one value, a new entity is added for each of these values. Each attribute to which no value has been assigned is set to a predefined

default value. If new entities are added to the simulation model, any attribute's value of the newly added entities can be accessed and stored in a *list*. For example, when new SKUs are added to a site, their *ids* can be accessed and used for further processing.

Entities can be removed from the simulation model by using the *remove statements*. Each remove statement is assigned to a single entity class that is specified during its implementation. Basically, all entities of a given entity class are affected by such a remove statement. However, if a subset of entities is to be removed, this set can be specified by using a *filter*, e.g., remove all SKUs from a location that weigh more than 100 kilogram. Similar to an add statement, certain attribute values of the removed entities can be accessed, for example the *ids* of the removed SKUs. These values can be stored in a *list* and used in the further course of the action type.

In order to ensure the requirement of traceability and simplification of the modeling of action types, there is the possibility to access and to use existing action types for the modeling process of further action types via the *call statement*. For the integration of an existing action type in the model, a reference to the action type must be made using the corresponding action type's name. In addition, the parameters of the called action type must be set accordingly.

An action type may have a return value as its result, e.g., a *list* of values. These values can be returned via the *return statement*. The returned values can be used for further processing in the calling action type.

Sometimes it is necessary to cancel the execution of an action type under certain conditions, e.g., if specific requirements are not fulfilled. For this purpose, the *abort statement* can be used. As a result of this statement, any changes made in the course of the same action type or by any action type called are undone.

6.5 Exemplary Implementation of the Action Type "Change Stock"

In the following, an exemplary implementation of the action type "Change Stock" is given and described. First, the *name* and *description* of the action type are defined. Then, two *filter* variables and one *modifier* variable, mySKUsFilter, mySitesFilter, and myStock, are declared as the action type's *input*, whereby the default value of the *modifier* variable is set to +10. The list of *statements* contains a *change statement* affecting the entity class SitesHaveSkus, in which inventory information of the underlying WLN is stored. The actual stock level of an SKU in a site is stored in the attribute Stock, whose value is changed by the *modifier* myStock. The affected entities are specified using the filter criteria of mySKUsFilter and mySitesFilter. Autonomous execution of derived actions is possible.

```
ACTIONTYPE   Change Stock
DESCRIPTION  This Action Type changes the Stock (myStock) of one or more SKUs
             (mySKUsFilter) in one or more Sites (mySitesFilter)
INPUT
   mySKUsFilter TYPE FILTER
   mySitesFilter TYPE FILTER
   myStock TYPE MODIFIER = +10
STATEMENTS
   IN TABLE SitesHaveSkus CHANGE
      Stock = myStock
   WHERE
      mySKUsFilter AND mySitesFilter
AUTONOMOUS TRUE
```

7 CONCLUSION AND OUTLOOK

The authors presented a method for modeling actions in data-driven, discrete event simulation models for WLNs. An essential part of this method is a semantic model for the description of action types. Using a domain-specific language for modeling action types in WLNs, a modeler can initialize the underlying

semantic model accordingly. The typical character of a DSL is achieved by the strongly reduced set of language constructs, new domain-specific data types and functions as well as the natural read-and-write flow during the modeling process. The DSL presented was used by domain experts to model exemplary action types in a materials trading company. The modeling process was simplified and accelerated.

Further research could be the extension of the semantic model by additional properties and of the language by corresponding constructs, e.g., boundary conditions relevant to planning, such as personnel or spatial capacities. A transfer of the concept to other domains would be an exciting option as well. In addition, the application of the developed DSL to other companies could lead to a broader validation.

ACKNOWLEDGMENTS

Special thanks to the Graduate School of Logistics, TU Dortmund and thyssenkrupp AG for supporting this research.

REFERENCES

Barth, K., M. Hartmann, and H. Schröder. 2015. *Betriebswirtschaftslehre des Handels*. 7th ed. Wiesbaden: Springer Gabler. https://dx.doi.org/10.1007/978-3-8349-7184-5.

Bentley, J. 1986. "Programming Pearls: Little Languages". *Communications of the ACM* 29(8):711–721.

Blutner, D., S. Cramer, S. Krause, T. Mönks, L. Nagel, A. Reinholz, and M. Witthaut. 2007. "Assistenzsysteme für die Entscheidungsunterstützung". In *Große Netze der Logistik*, edited by P. Buchholz and U. Clausen, 241–270. Berlin, Heidelberg: Springer.

Bockholt, F., W. Raabe, and M. Toth. 2011. "Logistic Assistance Systems for Collaborative Supply Chain Planning". *International Journal of Simulation and Process Modelling* 6(4):297–307. https://doi.org/10.1504/IJSPM.2011.048010.

Cheng, B. H. C., B. Combemale, R. B. France, J.-M. Jézéquel, and B. Rumpe. "On the Globalization of Domain-specific Languages". In *Globalizing Domain-Specific Languages*, edited by B. H. C. Cheng, B. Combemale, R. B. France, J.-M. Jézéquel, and B. Rumpe, 1–6. Springer, Cham. https://dx.doi.org/10.1007/978-3-319-26172-0

Cordeau, J.-F., F. Pasin, and M. M. Solomon. 2006. "An Integrated Model for Logistics Network Design". In *Annals of Operations Research* 144 (1): 59–82. https://doi.org/10.1007/s10479-006-0001-3.

Fowler, M. and R. Parsons. 2011. *Domain-specific Languages*. Upper Saddle River, New Jersey: Addison-Wesley.

France, R., and B. Rumpe. 2007. "Model-driven Development of Complex Software: A Research Roadmap". In *Future of Software Engineering*, edited by L. C. Briand and A. L. Wolf, 37–54. Los Alamitos, CA.: IEEE Computer Society.

Gutenschwager, K., M. Rabe, S. Spieckermann, and S. Wenzel. 2017. *Simulation in Produktion und Logistik. Grundlagen und Anwendungen*. Berlin: Springer Vieweg. https://doi.org/10.1007/978-3-662-55745-7.

Heilala, J., J. Montonen, P. Jarvinen, S. Kivikunnas, M. Maantila, J. Sillanpaa, and T. Jokinen. 2010. "Developing Simulation-based Decision Support Systems for Customer-driven Manufacturing Operation Planning". In *Proceedings of the 2010 Winter Simulation Conference*, edited by B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, 3363–3375. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Juan, A. A., J. Faulin, S. E. Grasman, M. Rabe, and G. Figueira. 2015. "A Review of Simheuristics: Extending Metaheuristics to Deal with Stochastic Combinatorial Optimization Problems". *Operations Research Perspectives* 2:62–72. https://doi.org/10.1016/j.orp.2015.03.001

Kengpol, A. 2008. "Design of a Decision Support System to Evaluate Logistics Distribution Network in Greater Mekong Subregion Countries". *International Journal of Production Economics* 115(2): 388–399. https://doi.org/10.1016/j.ijpe.2007.10.025.

Larson, P. D., and A. Halldorsson. 2004. "Logistics Versus Supply Chain Management: An International Survey". *International Journal of Logistics Research and Applications* 7(1):17–31. https://doi.org/10.1080/13675560310001619240.

Ma, H. and C. Suo. 2006. "A Model for Designing Multiple Products Logistics Networks". International Journal of Physical Distribution & Logistics Management 36(2):127–135. http://doi.org/10.1108/09600030610656440.

Nardi, B. A. 1993. A Small Matter of Programming – Perspectives on End User Computing. Cambridge, Massachusetts: MIT Press.

Oracle Corporation. 2019. *MySQL 8.0 Reference Manual*. https://docs.oracle.com/cd/E17952_01/mysql-8.0-en/mysql-8.0-en.pdf, accessed 12th June.

Rabe, M., M. Ammouriova, and D. Schmitt. 2018a. "Utilizing Domain-specific Information for the Optimization of Logistics Networks". In *Proceedings of the 2018 Winter Simulation Conference*, edited by M. Rabe, A. A. Juan, N. Mustafee, A. Skoogh, S. Jain and B. Johansson, 2873-2884. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Rabe, M., D. Schmitt, and M. Ammouriova. 2018b. "Improving the Performance of a Logistics Assistance System for Materials Trading Networks by Grouping Similar Actions". In *Proceedings of the 2018 Winter Simulation Conference*, edited by M.

Rabe, A. A. Juan, N. Mustafee, A. Skoogh, S. Jain and B. Johansson, 2861-2872. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Rabe, M., D. Schmitt, A. Klüter, and J. Hunker. 2019. "Decoupling the Modeling of Actions in Logistics Networks from the Underlying Simulation Data Model". *In Advances in Production, Logistics and Traffic. Proceedings of the 4th Interdisciplinary Conference on Production Logistics and Traffic 2019*, edited by U. Clausen, S. Langkau, and F. Kreuz, 32–44. Cham: Springer International Publishing.

Rushton, A. P. Croucher, and P. Baker. 2010. *The Handbook of Logistics & Distribution Management*. 4th ed. London, Philadelphia: Kogan Page.

Sammet, J. E. 1969. *Programming Languages: History and Fundamentals*. Englewood Cliffs, New Jersey: Prentice-Hall.

Schmitt, D. 2019. *Grammar of a Domain-specific Language for Modeling and Simulating Actions in Wholesale Logistics Networks*. http://www.itpl.mb.tu-dortmund.de/publikationen/files/Schmitt_DSL_Grammar_in_WholesaleLogisticsNetworks.pdf, accessed 12th June.

Seÿffert, R. 1972. *Wirtschaftslehre des Handels*. 5th ed. Opladen: Westdeutscher Verlag Opladen. http://doi.org/10.1007/978-3-322-83523-9.

SimPlan AG. 2017. "SimChain". http://www.simchain.net/, accessed 12th June.

Object Management Group. 2015. *Unified Modeling Language Specification*. https://www.omg.org/spec/UML/2.5.1/PDF, accessed 12th June.

van Deursen, A., P. Klint, J. Visser. 2000. "Domain-specific Languages: An Annotated Bibliography". SIGPLAN Not. 35(6):26–36. New York: ACM. http://doi.org/10.1145/352029.352035.

Voelter, M. 2010 – 2013. *DSL Engineering. Designing, Implementing and Using Domain-specific Languages*. Lexington, KY: CreateSpace Independent Publishing Platform.

Wang, N., Y. Mi, H. Gao, and W. Liu. 2016. "Logistics Network Model Based on Matter Element Node". *Procedia Computer Science* 91: 351–356. http://doi.org/10.1016/j.procs.2016.07.093.

Wenzel, S., B. Bockel, and F. Deist. 2009. "Die Integration der Produktions- und Logistiksimulation in die Digitale Fabrik – Herausforderungen und Entwicklungstrends". In *Digital Engineering -– Herausforderung für die Arbeits- und Betriebsorganisation*, edited by M. Schenk, 317–339. Berlin: GITO mbH Verlag.

Wexelblat, R. L. 1981. *History of Programming Languages*. New York: Academic Press.

AUTHOR BIOGRAPHIES

MARKUS RABE is a full professor for IT in Production and Logistics (ITPL) at the Technical University Dortmund. Until 2010 he had been with Fraunhofer IPK in Berlin as head of the corporate logistics and processes department, head of the central IT department, and a member of the institute direction circle. His research focus is on information systems for supply chains, production planning, and simulation. Markus Rabe is vice chair of the "Simulation in Production and Logistics" group of the simulation society ASIM, member of the advisory board of the Journal of Simulation, member of several conference program committees, has chaired the ASIM SPL conference in 1998, 2000, 2004, 2008, and 2015, and was local chair of the WSC'2012 in Berlin as well as proceedings chair of the WSC'18 in Gothenburg. More than 190 publications and editions report from his work. His e-mail address is markus.rabe@tu-dortmund.de.

DOMINIK SCHMITT works as a researcher at the Graduate School of Logistics and the ITPL at the Technical University Dortmund. He holds a diploma in Computer Science from the Technical University Dortmund. He graduated with a diploma thesis on the development of a computer-based model for the representation of transformable production systems for scheduling algorithms. Since 2016 he focuses his research on how to make the modeling and simulation of actions in wholesale logistics networks more accessible for non-simulation experts. His e-mail address is dominik.schmitt@tu-dortmund.de.