# Design of a Domain-Specific Language for Material Flow Analysis using Microsoft DSL tools: An Experience Paper

Bahram Zarrin
Denmark Technical University
baza@dtu.dk

Hubert Baumeister
Denmark Technical University
huba@dtu.dk

## Abstract

Material Flow Analysis (MFA) is the procedure of measuring and assessing the mass flows of matter (solid waste, water, food...) and substances (carbon, phosphorus ...) within a process or a system for the period of time. In this paper we propose a Domain-Specific Language (DSL) to model MFA in a waste management modeling context. The result is that we integrate the DSL within a waste management modeling software called EASETECH and we show how the proposed DSL allows the domain experts to extend the software without involving of software developers. Our future work is to develop more features for the DSL and make it more general to work for other flow analysis contexts.

***Categories and Subject Descriptors*** D.3.3 [*Programming Languages*]: Specialized application languages

***General Terms*** Languages, Design

***Keywords*** Domain-specific modeling languages, Material Flow, MS DSL tools

## 1. Introduction

Domain-Specific Languages (DSLs) are languages which are specialized for a specific application domain. In recent years they have become mostly used to improve the productivity of software developers and the quality of a software [7, 9, 10]. In this paper we show another application of them which is to utilize a DSL by domain experts to extend a software in order to fulfill new requirements.

EASETECH[1] is a novel modeling tool for life-cycle assessment (LCA) within the waste-management domain. The tool analyses waste flows, environmental emissions and resource consumption from waste-management systems and delivers a comprehensive impact assessment in relation to photochemical ozone formation, ozone depletion, potential global warming, acidification, nutrient enrichment, etc. [2]. The software employs a model consisting of a set of catalogs and material processes in order to describe scenarios of a solid waste system. The catalogs contain all the information related to environmental exchanges, resource consumptions, and emissions to water, soil and air [3]. This information is required to calculate the Life Cycle Impact Assessment (LCIA) of a waste system. A material process in EASETECH can be either a template material process or a composite material process based on several of these templates. A combination of these material processes models a waste scenario.

At the moment the template material processes, which are the basis and fundamental elements in waste scenarios, have been implemented in C#. This makes it difficult for the researchers and domain experts to add a new template material process to the software. One of the objectives of our research is to design a domain-specific language for waste-management modeling in order to describe different aspects of these waste processes and replace the hard-coded library of EASETECH with the compiler of this DSL. This allows the domain experts to extend EASETECH without dealing with any software-development activity.

In this paper we propose a DSL to describe material flows within a material process and we implement the DSL based on Microsoft DSL tools. This paper is organized as follows: first we have a brief introduction to Material-Flow Analysis in Sect. 2, then we design the proposed DSL and explain its semantics in Sect. 3. Afterwards we discuss the implementation of the DSL, related technologies, simulation and integration of the DSL in EASETECH in Sect. 4. We consider the related work in Sect. 5 and conclude our work in Sect. 7.

## 2. Material-Flow Analysis

The objective of using material-flow analysis (MFA) is to evaluate the totality and consistency of material flows between inputs and outputs of a certain system or process. One of the methods used commonly to do MFA is material-flow networks (MFNs) which were introduced many years ago and have been used regularly for Life Cycle Assessment (LCA)[4]. A material-flow network for a process can be defined as a set of inputs, outputs, transformers and transitions and it can be modeled as a directed graph such that inputs, outputs and transformers are the nodes and transitions are its edges. Transformers can change the material specifications, while transitions only transfer a specific amount of a material from a source node to a target node.

In EASETECH, a material is defined as a set of one or more fractions (such as paper, plastic, etc.) where each fraction has a list of substance names, amounts, and units. Based on these definitions, two type of material transformers are required to model a material process. One is needed to change the ratios of the fractions of a material and the other is required to change the ratios of the substances of a material. These material transformers can be modeled as a function that accepts a material object as input and generates a new material object as output.

## 3. Material Flow DSL

The meta-model of the DSL is illustrated in Fig. 1. In order to model the material flow in a material process, some material operators are defined in the meta-model. The model is a composite of one or more elements and each element is either a material element or a flow.

Three kinds of material elements are defined in the model, i.e. inputs, outputs, and operators. The model can have one or more inputs/outputs which will be mapped to the inputs/outputs of the material process. Operators are corresponded to material transformers in MFNs of which they can change the composition

[1] http://www.easetech.dk/

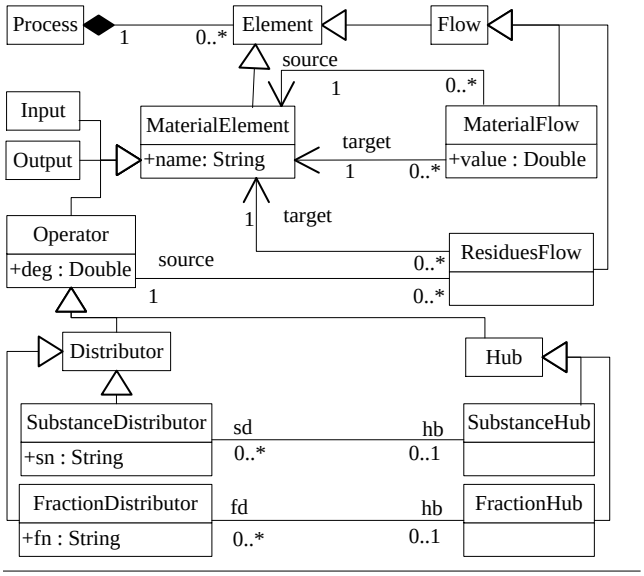


**Figure 1.** Meta Model of the Material Process model.

of the material. Two different operators are defined in the model. The first one is the Distributor, which can be either a Fraction Distributor (FD) or a Substance Distributor (SD). The other one is the Hub which can be either a Fraction Distributor Hub (FH) or a Substance Distributor Hub (SH).

FD operators are defined to extract a specific material fraction from the given material, while SD operators are proposed to extract a specific substance from the material. FD or SD operators can be directly used in a material-flow model or they can be hosted within a Hub. Hubs (FHs and SHs) are defined in the meta-model to be used in the material process wherever multiple substances or fractions are to be extracted and distributed from the given material.

Flows in the meta-model are corresponded to the material transitions in MFNs and they connect two elements in the model and transfer materials from their source elements to their target elements. Two different types of flows are defined in the model, i.e. Material Flow (MF) and Residues Flow (RF). The first flow operator transfers a portion of the material in the source element to the target element, while the other flow operator transfers the remaining material of the source element (which is a flow operator) to the target element.

### 3.1 Semantics of the Proposed DSL

In this section the formal definitions of materials is given first, and afterwards the formal semantics of a material process are presented and explained on the basis of these definitions.

#### 3.1.1 Formal Semantics of Material

As mentioned before, a material is a composition of material fractions (such as paper, plastic, etc.) and similarly a material fraction is a composite of different substances (such as water, Ca, Na, etc.). Therefore, in order to define a material, we need to define a material fraction first. To this end, we define *FN* and *SN* as a set of fraction names and substance names which can be used in material-process object diagrams (they are called catalogs in EASETECH). Then, based on these, we present the formal definition of a material fraction and a material accordingly.

A material fraction is defined as $f : SN \nrightarrow \mathbb{R}$, a partial function from substance names to its relevant amount of the substance in a fraction. The set of all material fractions is presented as $F$ and $f \in F$. The following arithmetic operators are defined over material fractions:

- The addition ($+ : F \times F \rightarrow F$) operator, merges two different material fractions.
- The subtraction ($- : F \times F \rightarrow F$) operator, subtracts a material fraction from another material fraction.
- The multiplication ($* : F \times \mathbb{R} \rightarrow F$) operator, rescales a material fraction.
- The filter operator ($|: F \times SN \rightarrow F$), filters the substances of a material fraction. The result is a material fraction with only one substance which has a same name as the right operand.

Based on the definition of material fraction and *FN* which is the set of fraction names, a material is formally defined as $m : FN \nrightarrow F$, a partial function from fraction names to material fractions. The set of all materials is presented as $M$ and $m \in M$. According to this definition and the arithmetic operators defined over material fractions, the following arithmetic operators are defined on material objects:

- The addition ($+ : M \times M \rightarrow M$) and the subtraction ($- : M \times M \rightarrow M$) operators are respectively defined over materials to merge two different materials or subtract a material from another material. In the same way,
- The multiplication ($* : M \times \mathbb{R} \rightarrow M$) operator overloaded to allow for rescaling a material.
- The filter operator ($|: M \times SN \rightarrow M, |: M \times FN \rightarrow M$) is similarly defined in order to create filters on fractions or substances of a material. This operator is used to extract specific substances or specific fractions from a material.

#### 3.1.2 Formal Semantics of Material Process

A formal definition of a material process ($P$) is presented as follows:

$$P = (I, T, E, O) \tag{1}$$

Where $I$ and $O$ is the set of inputs and the set of outputs. $T$ is the set of transition elements which can change the quantity of a material. $E$ is the set of transformer elements which can change the content of a material. ($I, O, T$ and $E$ are disjoint sets.)

Transition elements are directed arcs and they connect the transformers to each other and transfer materials between them. We identify the two ends of a transition $t \in T$ by writing $\uparrow t$ as the source of the transition and $\downarrow t$ as the target of transition, with the understanding that material moves from $\uparrow t$ to $\downarrow t, \uparrow t \in I \cup E, \downarrow t \in E \cup O$ and $\uparrow t \neq \downarrow t$.

According to the meta-model defined for the DSL, different material transformers are defined. Therefore, the set $E$ of transformers is the disjoint union (denoted $\uplus$ ) of four sets: the set $E_{FD}$ of fraction distributors, the set $E_{SD}$ of substance distributors, the set $E_{FH}$ of fraction hubs, and the set $E_{SH}$ of substance hubs:

$$E = E_{FH} \uplus E_{SH} \uplus E_{FD} \uplus E_{SD} \tag{2}$$

The following functions are defined in order to assign different attributes to the different kinds of transformers: $\texttt{deg} : E \rightarrow \mathbb{R}$ , is a function that assigns a real as degradation value to $e \in E$.
$\texttt{sn} : E_{SD} \rightarrow SN$, is a function that assigns a substance name to each substance distributor in $E_{SD}$.
$\texttt{fn} : E_{FD} \rightarrow FN$, is a function that assigns a fraction name to each fraction distributor in $E_{FD}$.
$\texttt{hb} : E_{FD} \uplus E_{SD} \nrightarrow FH \uplus SH$, is a partial function which specifies the hub that uses the given distributor as a port.
$\texttt{sd} : E_{SH} \rightarrow \mathcal{P}(E_{SD})$, is a function that assigns a set of substance

distributors as ports to each substance hub in $E_{SH}$.
$\texttt{fd} : E_{FH} \rightarrow \mathcal{P}(E_{FD})$, is a function that assigns a set of fraction distributors as ports to each fraction hub in $E_{FH}$.

According to the meta-model, the set $T$ of transitions is the disjoint union of two sets: the set $T_{MF}$ of material flows, and the set $T_{RF}$ of residues flows:

$$T = T_{MF} \uplus T_{RF} \tag{3}$$

A function, $\texttt{value} : T_{MF} \rightarrow \mathbb{R}$, is defined to assign a real value as the amount to the material flows. This value specifies the percentage amount of the material which the flow transfers from its source to its target. This value is undefined for residues flows.

In order to give semantics to the DSL, the following semantic functions for each syntactic category in process $P$ for given material input $I_0 : I \rightarrow M$ are defined as follows:

- $[\![\_]\!]^{\mathcal{I}} : I \times (I \rightarrow M) \rightarrow M$, determines the material value for an input element.
- $[\![\_]\!]^{\mathcal{O}} : O \times (I \rightarrow M) \rightarrow M$, determines the material value for an output element.
- $[\![\_]\!]^{\mathcal{E}} : E \times (I \rightarrow M) \rightarrow M$, calculates the transformed material by a transformer.
- $[\![\_]\!]^{\mathcal{T}} : T \times (I \rightarrow M) \rightarrow M$, calculates the material value transferred by a material transition.

Two more semantic functions $[\![\_]\!]^{\mathcal{E}}_{in}$ and $[\![\_]\!]^{\mathcal{E}}_{out}$ need to be defined in order to calculate $[\![\_]\!]^{\mathcal{E}}$. The first function evaluates the total material transferred into a material transformer by a set of transitions. The second function calculates the total material transferred out from a transformer through a set of transitions. Based on these semantic functions, we can define the semantic equations as follows:

For each input ($i \in I$), the evaluated material is the value assigned to $i$ in the given material input.

$$[\![i]\!]^{\mathcal{I}}(I_0) = I_0(i) \tag{4}$$

For the material transitions *MF*, the value is the percentage of the transformed material specified by its source element, while this value for the residues flows *RF* is the subtraction of the transformed material and the total material output of its source:

$$\begin{aligned} [\![mf]\!]^{\mathcal{T}}(I_0) &= \tfrac{value(mf)}{100} * \begin{cases} [\![\uparrow mf]\!]^{\mathcal{I}}(I_0), & \uparrow mf \in I \\ [\![\uparrow mf]\!]^{\mathcal{E}}(I_0), & \uparrow mf \in E \end{cases} \\ [\![rf]\!]^{\mathcal{T}}(I_0) &= [\![\uparrow rf]\!]^{\mathcal{E}}(I_0) - [\![\uparrow rf]\!]^{\mathcal{E}}_{out}(I_0) \end{aligned} \tag{5}$$

The total material input, $[\![\_]\!]^{\mathcal{E}}_{in}$, for each material transformer, if the transformer is a distributor and it belongs to a hub, is the material value of its hub $[\![\_]\!]^{\mathcal{E}}$, otherwise it is the sum of all the material transferred to the transformer by the transitions.

$$[\![e]\!]^{\mathcal{E}}_{in}(I_0) = \begin{cases} [\![hb(e)]\!]^{\mathcal{E}}(I_0), & e \in E_{FD} \cup E_{SD} \wedge hb(e) \neq \perp \\ \sum\limits_{t \in T \wedge \downarrow t = e} [\![t]\!]^{\mathcal{T}}(I_0), & else \end{cases} \tag{6}$$

The total material output, $[\![\_]\!]^{\mathcal{E}}_{out}$, for each fraction distributor and substance distributor is defined as follows:

$$\begin{aligned} [\![fd]\!]^{\mathcal{E}}_{out}(I_0) &= \sum_{t \in T \wedge \uparrow t = fd} [\![t]\!]^{\mathcal{T}}(I_0) \\ [\![sd]\!]^{\mathcal{E}}_{out}(I_0) &= \sum_{t \in T \wedge \uparrow t = sd} [\![t]\!]^{\mathcal{T}}(I_0) \end{aligned} \tag{7}$$

The total material output, $[\![\_]\!]^{\mathcal{E}}_{out}$, for each fraction hub or substance hub is the sum of material outputs of their distributors;

$$\begin{aligned} [\![fh]\!]^{\mathcal{E}}_{out}(I_0) &= \sum_{fd \in fd(fh)} [\![fd]\!]^{\mathcal{E}}_{out}(I_0) \\ [\![sh]\!]^{\mathcal{E}}_{out}(I_0) &= \sum_{sd \in sd(sh)} [\![sd]\!]^{\mathcal{E}}_{out}(I_0) \end{aligned} \tag{8}$$

The semantic equations for the material transformers are defined as follows:

$$\begin{aligned} [\![fh]\!]^{\mathcal{E}}(I_0) &= \tfrac{100-deg(fh)}{100} [\![fh]\!]^{\mathcal{E}}_{in}(I_0) \\ [\![fd]\!]^{\mathcal{E}}(I_0) &= \tfrac{100-deg(fd)}{100} [\![fd]\!]^{\mathcal{E}}_{in}(I_0)\,|_{fn(fd)} \\ [\![sh]\!]^{\mathcal{E}}(I_0) &= \tfrac{100-deg(sh)}{100} [\![sh]\!]^{\mathcal{E}}_{in}(I_0) \\ [\![sd]\!]^{\mathcal{E}}(I_0) &= \tfrac{100-deg(sd)}{100} [\![sd]\!]^{\mathcal{E}}_{in}(I_0)\,|_{sn(sd)} \end{aligned} \tag{9}$$

The semantic function for output elements is defined as follows:

$$[\![o]\!]^{\mathcal{O}}(I_0) = \sum_{t \in T \wedge \downarrow t = o} [\![t]\!]^{\mathcal{T}}(I_0) \tag{10}$$

Based on these semantics functions, we can give semantics to a material process $P$ as well. Since the purpose of the material process is to calculate the outputs of the process based on given inputs, then the semantic function for a process $P$ and given input, $I_0 : I \rightarrow M$, is defined as follows:

$$\begin{aligned} &[\![P]\!] : P \times (I \rightarrow M) \rightarrow (O \rightarrow M) \\ &[\![P]\!](I_0) = \lambda o : O.[\![o]\!]^{\mathcal{O}}(I_0) \end{aligned} \tag{11}$$

# 4. Implementation

The Visualization and Modeling SDK (VMSDK) is used to build powerful domain-specific development tools which can be integrated into Microsoft Visual Studio. In the core of VMSDK is the model definition (meta-model) of the DSL used to symbolize the concepts of the specific domain. The model is surrounded by a range of tools, such as a diagram editor, code generator, APIs to interact with the IDE of Visual Studio, etc. In the following section, the implementation of the proposed DSL based on VMSDK is explained.

## 4.1 Syntax of the Proposed DSL

The DSL definition diagram for the proposed DSL in Visual Studio is illustrated in Fig. 2.As presented, a domain class called *Material Process* is used as the root element of the diagram which represents the model. Two abstract domain classes called *Element* and *MaterialElement* derived from *Element* are used in the model as the base classes. The *Element* class is used as the base class for all of the model elements and the *MaterialElement* class is the base class for all of the material elements in the model.

An abstract domain class called *MaterialOperator* derived from *MaterialElement* is defined as the base class for all of the operators that can be defined for a material process. Another abstract domain class called *FlowOperator* is added to the diagram to be used as the base class for all of the material flow operators that are dealing with material transformation. This class is derived from *MaterialOperator*. Two abstract domain classes called *Hub* and *Distributor* which are inherited from *MaterialOperator* are defined in the model as the base class for Hub and Distributor operators.

According to the meta-model, for each different type of the hub and distributor operators (Fraction Hub, Substance Hub, Fraction Distributor and Substance Distributor), a related domain class derived from the related abstract class is defined and added to the diagram. Finally, for all of the non-abstract domain classes, which should appear as an element in the DSL diagram, an embedding relationship is defined between the domain class and *MaterialProcess*. A shape class is also defined for each of them and mapped to

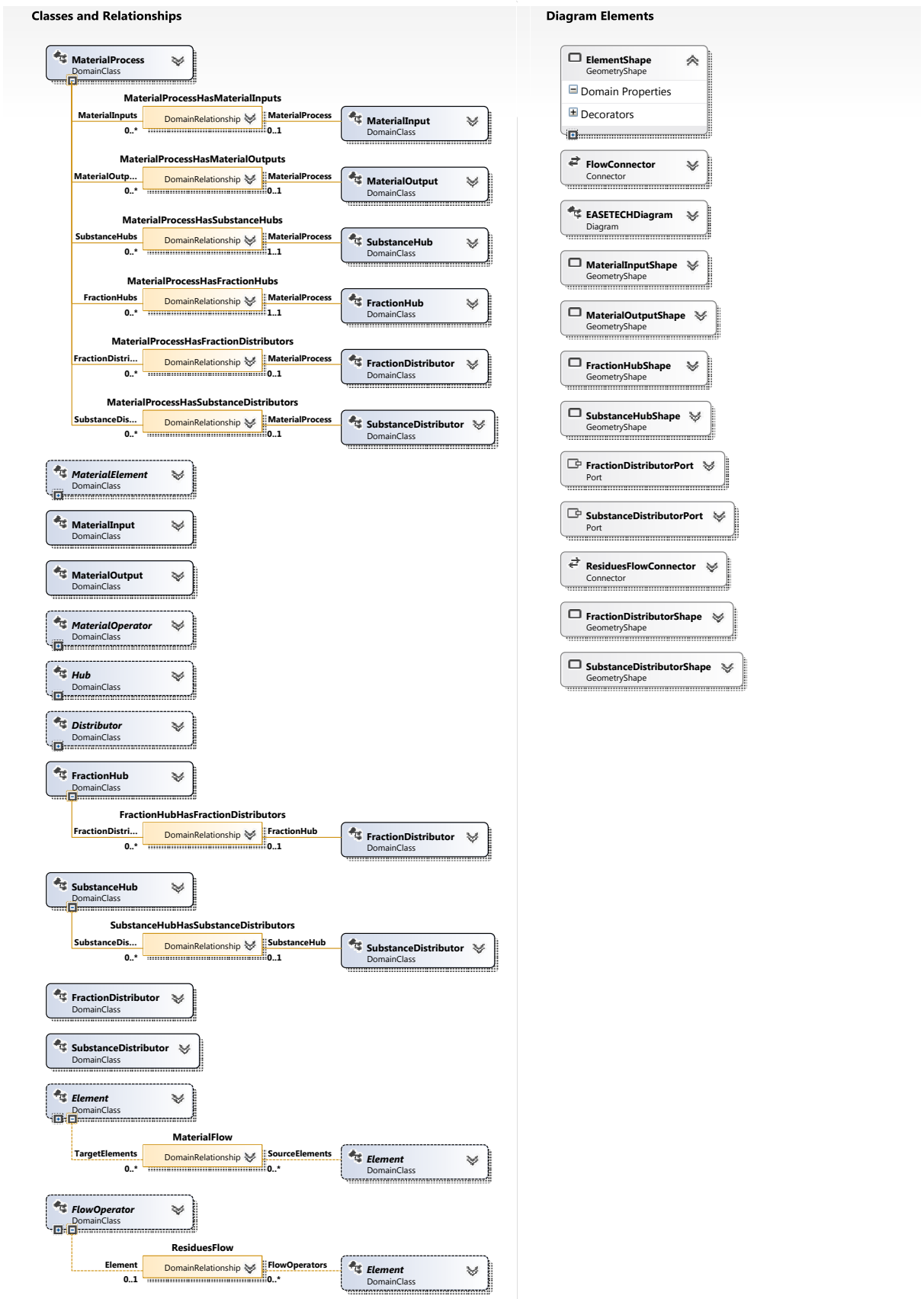


**Figure 2.** Meta Model of the Material Process model. This diagram is modified in order to fit in one page.

the domain class to describe the concrete syntax of the operator in the model diagram.

In order to provide connectivity between the model elements and show the material-transformation flow between the material operators, two reference relationships are defined. The first relationship is defined between *Element* and itself, which allows each element of the diagram to be linked to the other elements. This reference relationship is called *MaterialFlow* and is described by means of a domain relationship class. The relationship class has one property called *value* which specifies the percentage amount of the material that should be transferred from the source element to the target element of the relation. The other reference relationship called *ResiduesFlow* is defined between *FlowOperator* and *Element*, which allows one to transfer the residues material from a flow operator to any element in the diagram. To represent the links between the elements in the model diagram, for each reference relationship in the model a connector shape class is created and mapped to the relation.

### 4.2 Semantics of the Proposed DSL

This section describe the implementation of the static and dynamic semantics of the proposed DSL. There are two types of links in the model; *MaterialFlow* and *ResiduesFlow*. Both of these links should be validated regarding the type of the elements which can be used as source or target of the link. The accepting element types for source and target of these links are listed in the following table.

**Table 1.** Valid connectivity table between the elements

| | Material Flow | | Residues Flow | |
|---|---|---|---|---|
| Element Types | Source | Target | Source | Target |
| Input | X | | | |
| Output | | X | | X |
| Fraction Hub | | X | X | X |
| Substance Hub | | X | X | X |
| Fraction Distributor | X | X | X | X |
| Substance Distributor | X | X | X | X |
| Aggregator | X | X | | X |

The DSL designer provides some mechanisms to allow the developers to control and customize the creation or modification of the links. For each type of link defined for the model, a connection builder node will be created automatically. In order to apply customization to the link, an object called Link Connect Directive can be used that allows the DSL developers to define the valid types for source and target of the Link.

In order to apply the constraints in the table to the links, for each link, a Link Connect Directive is defined according to the above constraints, and added to the related connection builder in the designer. The other static semantics related to validating the value of the properties for the diagram elements are done through extending the domain classes with partial classes and writing a custom validation function for them.

In order to implement the dynamic semantics explained in Sect. 3.1, The generated codes for the proposed DSL are extended. To achieve this goal, a material property is added to the *MaterialElement* domain class which is the base class for all the material elements of the model. This property is defined as a calculated property for the domain class, which means that the DSL designer will generate a *GetMaterialVlaue* in the generated code for the domain class and expects this method to be implemented in the related partial class. A partial class is created for each domain class in the DSL definition, and since they are inherited from *MaterialElement*, they can override *GetMaterialVlaue* with their own calculation according to the semantic functions $[\![\_]\!]^{\mathcal{I}}$, $[\![\_]\!]^{\mathcal{O}}$ and $[\![\_]\!]^{\mathcal{E}}$ explained in Sect. 3.1.

### 4.3 Simulation

To enable the user to generate material in order to simulate the model, two properties are added to the *MaterialInput* domain class. The first property is the amount of the material that should be generated, and the second property is a list of the fractions which should be included in the material composition. On the basis of these properties, a material will be generated and considered as the process input for the simulation.

Two different views are created to visualize the material composition and material generation. Another view called *MaterialView* is created, which surrounds the other views and swaps the views based on the selected element in the diagram. If the selected element is a material input element, it shows the material-generation view, otherwise it shows the material-composition view. In order to show the material view for the selected element in Visual Studio, *MaterialFlowWindow* class is defined. This class creates a tool window in Visual Studio IDE and shows the *MaterialView* whenever a DSL diagram is loaded in Visual Studio.

### 4.4 Code Generation

For generating codes from the proposed DSL model, Microsoft Text Template Transformation Toolkit (T4 template) is used. Two different types of code are generated from the proposed DSL model. The first type is the generated code to do the material calculation for each element in the model, and the other type is the generated code used to add a material process template to the EASETECH process library.

#### 4.4.1 Generating Code for Material Calculation

In order to generate material-calculation code, a T4 template is defined for each domain class in the DSL definition. The template generates a class for each element in the model which has the same type as the domain class of the T4 template. The template also adds the following functions to the classes; *GetMaterialInputs*, *GetMaterialValue*, *GetResiduesMaterial* and *GetMaterialOutputs*. The implementation and availability of these functions are generated according to the type of the element and the semantic of the element, which is explained in Sect. 3.1.

#### 4.4.2 Integrating with EASETECH

In order to make the generated code simple and reusable, some base classes are defined in a new assembly, which is called *EASETECH.DSL.Lib*. These classes are TCMaterialProcessTemplate, MaterialProcessTemplate and MaterialOutputTemplate, and are derived from the related classes in EASETECH. They are implementing some basic functionalities which are required by the generated classes. To generate a material process template which can be used in EASETECH, two T4 templates are defined on the basis of an instance of the proposed model. One of them, called *TCTableCodeGenrator*, is used to generate a class based on *TCMaterialProcessTemplate* that is responsible for material calculation of the material-process outputs. The other template, which is called *MaterialProcessCodeGenerator*, is used to generate a material-process class based on *MaterialProcessTemplate*. Whenever an instance of the DSL is compiled, an assembly will be generated to be used in EASETECH.

At the end, some changes have been applied in the loading method of the material-processes library in EASETECH to import the generated material process. The loading method has been changed in such away that it dynamically loads all the assemblies which are generated from the instances of the proposed model. After that, it adds all the types within these assemblies which are based on *MaterialProcessTemplate*, to the material processes library.

## 5. Related Work

In recent years, material-flow networks [6] have been known as one of the appropriate methods of doing MFAs [5]. Different tools and approaches have been proposed to model and simulate MFA within different contexts, and the most relevant of these are mentioned in this section. Umberto was developed in 1997 as an initial material-flow analysis tool[8]. This tool is one of the powerful material-flow analysis tools and it provides interfaces to other programs. It also allows the users to extend the transitions based on their needs by using Microsoft Active Scripting. In 2006, the Vienna University of Technology developed a freeware software for MFA called STAN (short for subSTance flow ANalysis), which supports MFA according to the Austrian Standard ONORM S 2096 and allows consideration of data uncertainties [1]. Unlike the mentioned tools, a component-based approach to MFA is presented in [11] which integrates material-flow analysis and discrete event simulation into a component-based framework to ease both model development and maintenance.

In comparison of our work with the related work, most of these approaches offer a generic tool for material-flow analysis, which has been developed based on non-model-driven approaches. In contrast, we propose a specific material-flow analysis tool in the context of waste-management, and we use a model-driven and language-oriented approach to address the problem.

## 6. Results and Discussions

On the basis of this experience, we found that DSL tools are mature enough to develop a complete DSL project. VMSDK provides a special editor to describe a meta-model together with a graphical notation for a DSL. It generates a strongly typed implementation of the domain classes for the model, which runs in a transaction-based store, a model explorer and a diagram editor, serialization objects which store the models in XML format, and mechanisms for generating code or other artifacts from the model by using text templates. All the generated features can be customized and extended in a way that still allows the developers to update their DSL definition and regenerate the features without losing their extensions and customization code.

One of the drawbacks of implementing DSLs based on this framework is the lack of support to formalize the semantic of the DSL, which led us to implement the DSL semantic in an informal way twice for simulation and code-generation purposes. This made the semantic verification and maintenance of the DSL more difficult.

The other problem we found in this experience is the visualization of the meta-model for a DSL which is presented in Fig. 2. Although the mapping between the abstract syntax and concrete syntax of the DSL is presented well here, understanding the meta-model of the DSL in this diagram, compared to the meta-model diagram in EMF (Fig. 1), is more difficult, especially when the DSL definition is more complex.

While in other frameworks, like EMF, the meta-model of a DSL can be reused to design different types of DSL (such as textual or graphical languages), DSL tools can only be used to develop graphical languages, and the DSL definition cannot be reused to develop textual languages.

## 7. Conclusions

In this work we proposed a Domain-Specific Language for Material-Flow Analysis with a stand-alone tool support. This DSL can help researchers to model and simulate material flows of a material process. We also shared our experience in developing DSLs with Microsoft DSL tools.

In addition, in this paper we showed that, thanks to DSL technologies, a software like EASETECH can be extended with new requirements directly by the domain experts. Before this, the environmental scientists had to ask the developers of EASETECH to add new contributions or requirements to the software. Now, by using this DSL, they will be able to do it themselves.

Our future work will develop more features for the DSL and make it more general to work for other flow analysis contexts.

## References


[1] O. Cencic and H. Rechberger. Material flow analysis with software STAN. In M. S. E. Andreas Moeller, Bernd Page, editor, *Shaker Verlag*, pages 440–447. Shaker Verlag, 2008.

[2] T. Christensen, G. Bhander, H. Lindvall, A. Larsen, T. Fruergaard, A. Damgaard, S. Manfredi, A. Boldrin, C. Riber, and M. Hauschild. Experience with the use of LCA-modelling (EASEWASTE) in waste management. *Waste Management and Research*, 25:257–262, 2007.

[3] J. Kirkeby, H. Birgisdottir, T. Hansen, T. Christensen, G. Bhander, and M. Hauschild. Environmental assessment of solid waste systems and technologies: EASEWASTE. *Waste Management and Research*, 24(1):3–15, 2006.

[4] H. Lambrecht and M. Schmidt. Material flow networks as a means of optimizing production systems. *Chemical Engineering and Technology*, 33(Issue 4):610–617, 2010.

[5] H. Lambrecht and M. Zimmermann. Combination of optimization methods and material flow analysis for improvement of operational material use (KOMSA): Concept and its implementation. In M. S. E. Andreas Moeller, Bernd Page, editor, *Shaker Verlag*, pages 310–318. Shaker Verlag, 2008.

[6] L. e. a. Möller A., Stoffstromnetze. In Hilty. Material flow networks as a means of optimizing production systems. *Informatik und Umweltschutz*, 33(Band 2):610–617, 1994.

[7] A. L. Santos, K. Koskimies, and A. Lopes. Automating the construction of domain-specific modeling languages for object-oriented frameworks. *Journal of Systems and Software*, 83(7):1078 – 1093, 2010.

[8] H. Schmidt, Möller and Beilschmidt. Environmental material flow analysis by network approach. In *11th International Symposium of the German Society for Computer Science (GI)*, pages 768 – 779. Umweltinformatik, 1997.

[9] B. Selic. Personal reflections on automation, programming culture, and model-based software engineering. *Automated Software Engineering*, 15(3-4 SPEC. ISS.):379–391, 2008.

[10] M. C. Viana, R. A. Penteado, and A. F. do Prado. Domain-Specific Modeling Languages to improve framework instantiation. *Journal of Systems and Software*, 86(12):3123 – 3139, 2013.

[11] V. Wohlgemuth, B. Page, and W. Kreutzer. Combining discrete event simulation and material flow analysis in a component-based approach to industrial environmental protection. *Environmental Modelling and Software*, 21(11):1607 – 1617, 2006.