Mathematical Modelling of Systems
1996, Vol. 1, No. 1, pp. 000–111

1381-2424/96/0201-00$12.00
©Swets & Zeitlinger

Modelling unit processes using formal language description and object-orientation

S.O. WASBØ[*] AND B.A. FOSS[*]

ABSTRACT

In this paper we demonstrate how object-orientation and formal languages can be applied in modelling unit processes. We suggest a model representation where unit processes are decomposed into a two-level hierarchy based on a set of elementary building blocks. The formal language description is used as an alternative representation of the model. We show how it can be used for checking model consistency and as a basis to derive model equations. The possible use of the concept in a model assistant is discussed.

Key words: Process modelling, object-orientation, formal languages, directed graphs, modelling assistant.

1 INTRODUCTION

The process industries develop and use mathematical models for plant design and operations. Relatively large resources are used to develop these models. Viewing the situation in more detail, industrial companies tend to develop process models as an isolated activity in the sense that different modelling projects have a low degree of interaction. Hence, there is a tendency to start from 'scratch' every time there is a need for a new model. An important

[*]Department of Engineering Cybernetics, The University of Trondheim - NTH, 7034 Trondheim, Norway, Fax: +47-735-94399 / email: {Stein.O.Wasboe,Bjarne.Foss}@itk.ntnu.no

reason for this is the observation that a model depends *both* on the process as well as the application in question. Models for different applications will therefore differ.

There is a clear tendency towards a proliferation of models in applications like control, diagnosis and planning. This is caused by the simple fact that these types of applications usually improve performance. There are numerous successful examples of this throughout the literature. Two examples from the chemical industries in Norway are a nonlinear model-based controller of a polymerization reactor (Singstad 1992) and a diagnosis scheme for discharge surveillance in a fertiliser plant (Mjaavatten 1994).

In the process industries we may define two levels of models; plant models and models of unit operations such as reactors, pumps, heat exchangers, and tanks. There exists modelling tools to develop plant models on the basis of a library of models of unit operations, e.g. Eikaas (1990). These are flexible in the sense that unit models may be changed relatively easy. The key to this flexibility is the modularity of the plant model and the well-defined interface between the unit models. In this paper we focus on a methodology for supporting mathematical modelling of the unit processes themselves, focussing on modelling the topology and the phenomena taking place in the furnace rather than writing mathematical equations directly. The possibility of generating mathematical equations from this phenomenological structure is then examined.

The paper is organised as follows: First, the methodology is presented. This includes the choice and structure of the elementary building blocks, a discussion on the choice of generality of the building blocks, the use of object-orientation, and how a formal language description can be utilised to represent and analyse a model. An initial description of parts of this can be found in Wasbø and Foss (1995). Second, the methodology is applied to a semi-realistic example from the chemical process industries. Finally, we highlight the contribution of the paper in the conclusions.

2 METHODOLOGY

We start by proposing a set of elementary building blocks. Thereafter we argue that it is necessary to limit the domain of application in order to obtain efficiency in model development. Finally, we examine how object-oriented mechanisms and a formal language description of the dynamic model can be included to further improve model development efficiency.

2.1 Elementary building blocks

Our hypothesis is that model development of unit processes can be made more efficient by defining a set of elementary building blocks. We define two levels of building blocks. One topological and one phenomenological. Let the topological level consist of two sets: **ED** (elementary devices) and **EC** (elementary connections). The notation *devices* and *connections* is adopted from Marquardt (1994). An **ED**-element has in this context the ability to contain material and energy (i.e. a control volume of a chemical phase), while an **EC**-element represents transport of material and energy between **ED**-elements. The **ED** and **EC** sets contain in general a number of different types of devices and connections. A device can for instance be a liquid phase or a gas phase. Let $\mathcal{B}$ denote the set containing all **ED** and **EC** elements.

The phenomenological level contains three sets: **A** (accumulation), **T** (transport) and **R** (reactions). The **A**-elements are chemical species and energy, which are allowed to accumulate inside a device. The **T**-elements represent flow of the chemical species and energy between the devices. Elements of **A** can therefore only be found in **ED**-elements, while **T**-elements can only be found inside **EC**-elements. Elements from the **R**-set can be found in both devices and connections, since a reaction can take place either inside a volume or on a surface between two phases.

A model of a unit process, M, will consist of a network of volume and flux elements as shown in Figure 1. These model elements are instantiated from the **ED** and **EC** sets. At the extreme this network consists of only one volume element. This will be the case if we model a process as a single batch reactor, and are not interested in the dynamics related to the filling and tapping of the reactor. At the other extreme the network may consist of a large number of connected volume and flux elements. This will often be the case if we choose to approximate a distributed model by a finite number of states.

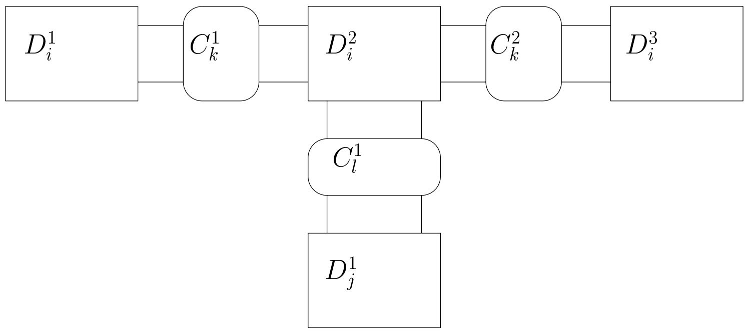

Fig. 1. The figure shows a possible coupling of elements, on a topological level, instantiated from **ED** and **EC**. The superscript is used to indicate the class the element belongs to.

The network is a directed graph where the nodes symbolise the volume elements and the flow elements. Double lines indicate a connection between an **EC**-element and an **ED**-element. The graph must be consistent. By this we mean that only allowable couplings should be possible. As an example the flux from one volume element will usually be of the same kind as the connecting volume element. There are exceptions to this, e.g. a flash valve where a liquid flow is transformed to a gas or gas/liquid flow due to a pressure drop.

There is a large degree of freedom in specifying the elements of **ED** and **EC**. They can be made very general, only containing knowledge of the volume and the connecting geometry. On the other hand, they can be specialised containing information like shape, type of phase(s), as well as process specific information. An example of a specialised D_i would be a tray in a distillation column.

Analogous to the model elements instantiated on the topological level, the model elements instantiated from **A**, **T** and **R** form a directed graph describing the interactions on a phenomenological level. This is shown in figure 2.

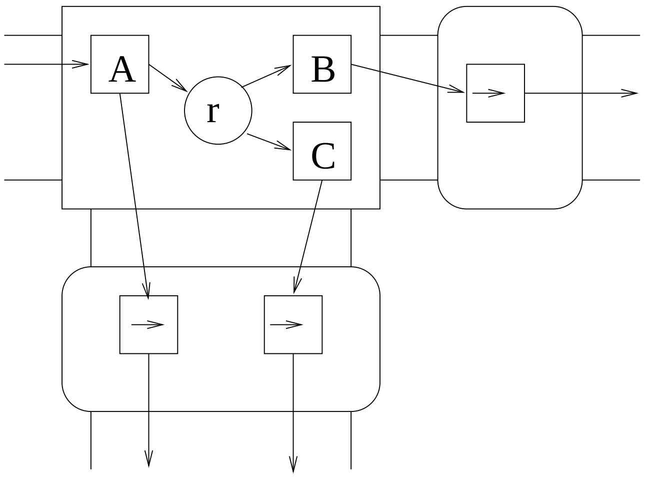

Fig. 2. phenomenological level network. The entities A, B and C are instantiated from **A**, the reaction r from the set **R**, and the transport symbols from **T**.

Note that the network on the phenomenological level closely resembles the topological level network since the topological level set **ED** relates to the set **EC** in the same manner as the phenomenological level set **A** relates to the sets **R** and **T**. Furthermore, between two elements from the set **ED** there is always an element from **EC**, and between two elements from **A** there is always an element from the sets **T** or **R**.

There are examples from the process industry which may seem to violate the structure above. For instance, two valves (connections) may be linked directly to each other. This configuration will, however, be modelled by adding a control volume between the valves, which means adding artificial dynamics, or by regarding the two valves as one unit, and solving algebraic equations to determine the two pressure drops within the unit. Hence, the model structure includes this case. Another example is the modelling of a plug flow reactor (PFR) as a sequence of control volumes. This gives several devices that are directly linked to each other. Nevertheless, there are flows given by some potentials between the control volumes. Hence, we have (artificial) connections between the volumes, and again our structure includes this case.

For a more detailed description of the graphical symbols, cf. Drengstig *et al.* (1996) or Wasbø (1996). In these works, the possibility of making composite devices and connections are also being discussed. Such composite structures are necessary for large systems.

2.2 The domain of application and the choice of building blocks

Our aim is to improve modelling efficiency by the proposed model structure. Two important choices are (i) the domain of application and (ii) the generality of the elements in the sets.

For a set of processes *and* a set of applications there will exist a set of relevant models $\mathcal{M}$. It is fairly straightforward to understand that the set $\mathcal{M}$ increases if the range of processes and/or the range of applications increase.

The generality of the elements within $\mathcal{B}$ will determine the class of models, $\mathcal{M}_{\mathcal{B}}$, that can be instantiated from $\mathcal{B}$. Obviously, we must make sure that $\mathcal{M} \subset \mathcal{M}_{\mathcal{B}}$. Making the elements more specialized limits the class of models. We advocate that both general as well as specialised elements should be included into $\mathcal{B}$. This is based on the following two observations:

- The advantage of general elements is that the possibility for reuse increases. Assume that we have two elements *control volume* and *gas-filled volume* within **ED**. The element *control volume* is obviously more general than *gas-filled volume.* We can e.g. use *control volume*, as opposed to *gas-filled volume*, to model some liquid reactor.

- If we want to model a gas-filled polymerization reactor it is simpler to build this using the element *gas-filled volume* than from the element *control volume*. The reason for this is that more needs to be specified when using *control volume* than *gas-filled volume* as a basis for modelling a gas-filled reactor.

The model $\mathcal{M}$ will describe the topology of devices and connections in the process (**ED** and **EC**-elements), *and* the internal topology of the components within the devices and connections. This topology will be sufficient to describe the main differential equations in the process. Detailed calculations concerning component fluxes and reactions, initial values and parameters should be provided from predefined objects, in the object-oriented model strategy, or by the modeller. The final model is linked to an equation solver for simulation.

2.3 Object-oriented model

In the preceding sections we have emphasised a highly modular approach to model process units. Object-orientation (OO) offers some useful features as an addition to standard modularisation. These are *classification*, *inheritance*, *encapsulation* and *polymorphism* (Winblad *et al.* 1990, Rumbaugh *et al.* 1991). Object-oriented modelling was first addressed by Elmqvist (1978) who developed a general modelling language for large continuous systems. This work has lead to the development of Omola (Nilsson 1993, Mattson and Andersson 1992).

In this work we use the concept of object-orientation to define model building blocks on two abstraction levels: 1) the topological level and 2) the phenomenological level. Hence, instead of defining models based on mathematical equations directly, we attempt to move focus from an equation-based modelling to a more phenomenological based modelling. At a certain degree of detail, however, the phenomena are described by mathematical equations.

2.4 Formal language description

The modular or object-oriented approach turns some of the modelling from time consuming studies and advanced implementation into configuring predefined elementary blocks. This configuring is, however, not necessarily straightforward. Some elementary blocks should not be connected, e.g. a gas valve between two liquid buffer tanks. A modelling assistant could be constructed to guide the modeller and omit obvious configuration errors. We propose to use a formal language description (FLD) of the configuration network as a means to aid the modelling process.

We use a formal language description to deal with strings and sets of strings. The definition of a formal language is given in appendix 4.

Conditions for production rules in process modelling

In the previous section we found that there are rules on what kind of elements can be connected. These rules strongly resemble the production rules P, for phrase-structure grammars. We try to utilise the above definition of a phrase-structure grammar in creating a new type of representation for process models.

Studying the nature of our system, the following grammatical rules seem to be valid for the topological level (the sets **ED** and **EC**):

$$< \text{String} > \to < \text{Device} >$$
$$< \text{String} > \to < \text{Device} >< \text{Connection} >< \text{String} >$$

The two rules indicate that elements of **ED** may stand alone, while elements of **EC** must be linked to at least two other elements from the set **ED**.

The grammatical rules for the phenomenological level is very similar:

$$< \text{String} > \rightarrow < \text{Accumulation} >$$
$$< \text{String} > \rightarrow < \text{Accumulation} >< \text{Transport} >< \text{String} >$$
$$< \text{String} > \rightarrow < \text{Accumulation} >< \text{Reaction} >< \text{String} >$$

As for the topological level, the two rules indicate that elements of **A** may stand alone, while elements of **T** and **R** must be linked to at least two other elements from the set **A**.

There is a large degree of freedom in the definition of the elements of **EC**, **ED**, **A**, **T** and **R**. The consequence of this is that *all* the elements of **ED** may not necessarily be compatible with *all* the elements of **EC**. Likewise for the sets **A**, **T** and **R**. This means that there are certain conditions that must be fulfilled before two elements can be linked. For example, for a valve and a reactor to be linked, they must be designed for the same kind of material (e.g. gas-valve and gas-reactor).

The following conditions apply for configuring a model on the topological level:

Condition 1 A connection may be linked to a device if the elements from **T** and **R** present in the connection can be linked to a subset of the elements of **A** in the device.

This condition imply that the **A**-elements instantiated in **ED** elements lead to constraints on how the **ED** and **EC** elements may interact.

It is obvious that since a connection describes a flow of a chemical species or energy, these entities must be present in the linked devices.

Condition 2 A connection must be linked to at least two devices.

A connection describes the transport between devices. The flow rate will typically depend on a potential between two devices.

The following condition applies for configuring a model on the phenomenological level:

Condition 3 A reaction must be linked to at least two A elements.

Condition 4 A transport must be linked to exactly two A elements.

In addition to the above we make the following definition:

Definition 1 The order of A, R and T elements in a string represents the positive direction of flow in the modelled system.

Context-free grammar

From the above rules we can define the following grammar for the topological level:

$$\mathcal{G} = (N, T, P, \sigma) \tag{1}$$

where

$$N = \{W_1, W_2, \ldots, W_r, \sigma\},$$
$$T = \{c_1^1, c_1^2, \ldots, c_1^s, c_2^1, c_2^2, \ldots, c_m^t, d_1^1, d_1^2, \ldots, d_n^u\}$$

The symbols $W_1, W_2 \ldots W_r$ represent intermediate stages in the production of a string. The number of such stages depends on the number of possible combinations of elements from **EC** and **ED**, fulfilling Condition 1. The set T includes all the connections and devices that are present in the application model. The c_k^vs are instantiations of connections from **EC** while the d_j^ws are instantiations of devices from **ED**. In other words, the T-set contains the actual models of the process (e.g. control-volume-number-3) , not types of models (e.g. control-volume).

The production rules are given by: $P = \{$

$$\sigma \to \lambda, \tag{2}$$
$$\sigma \to W_l \qquad l = 1 \ldots r, \tag{3}$$
$$W_i \to d_j c_k W_l \qquad i = 1 \ldots p,\ k = 1 \ldots m,\ j = 1 \ldots n,\ l = 1 \ldots p, \tag{4}$$
$$W_i \to d_j \qquad i = 1 \ldots p,\ j = 1 \ldots n \tag{5}$$

$\}$

where the indices i, j, k, l must be chosen so that Condition 1 above is fulfilled. A more specific grammar cannot be defined until the set of terminals, T, has been chosen in an application.

From the definitions earlier we see that the above grammar is a context free grammar. We will transform this to obtain a regular grammar since this has a close relationship to finite state automata which is a concept to be used later.

Regular grammar

We redefine $N = \{W_1, W_2, \ldots, W_p, V_1, V_2, \ldots, V_r, \sigma\}$, while T remains the same. The added symbols $V_1, V_2, \ldots, V_r$ represent another intermediate stage in the production. Again, the number of V_is will depend on how Condition 1 is fulfilled.

The production rules are changed to: $P = \{$

$$\sigma \to \lambda \quad , \tag{6}$$
$$\sigma \to d_j \qquad j = 1 \ldots n, \tag{7}$$
$$\sigma \to d_j V_l \qquad j = 1 \ldots n,\ l = 1 \ldots r, \tag{8}$$
$$V_l \to c_k W_i \qquad l = 1 \ldots r,\ k = 1 \ldots m,\ i = 1 \ldots p, \tag{9}$$
$$W_i \to d_j V_l \qquad i = 1 \ldots p,\ j = 1 \ldots n,\ l = 1 \ldots r, \tag{10}$$
$$W_i \to d_j \qquad i = 1 \ldots p,\ j = 1 \ldots n \tag{11}$$

$\}$ where the indices i, j, k, l must be chosen so that Condition 1 above is fulfilled. This is a regular grammar, which can be represented by a directed graph, a finite state automaton.

The first production rule ($\sigma \rightarrow \lambda$) indicates that we may define empty strings in the languages. The next two rules show the other possible start alternatives, where nonempty strings are produced. A string containing only one connection, or a string starting or ending with a connection is not allowed. Strings containing more than one symbol is produced by the fourth and fifth rule. The first and sixth rule terminate a string. Note that all strings in the language $L(\mathcal{G})$, except the empty string, will contain an odd number of symbols (1, 3, 5, 7, etc.).

This means that all symbols placed as odd numbers in a string represent devices. Symbols placed as even numbers in a string represent connections.

Our claim is that languages defined by such grammars, based on a small number of topological and phenomenological symbols can be used to define a large class of dynamic process models.

Constructing an automaton

An automaton is similar to a finite-state machine. The difference lies in the fact that an automaton has 'accepting' and 'rejecting' states rather than some output. Furthermore and important to us, a finite state automaton (FSA) is capable of executing algorithms for 'accepting' or 'rejecting' strings in a language $L(\mathcal{G})$ based on a regular grammar (Gill 1976). See appendix 4 for a definition of an FSA.

Again, using the notation from Gill (1976), we get an automaton as described in figure 3[1]. The finite state automaton can be used in consistency checks, rejecting or accepting strings.

The automaton is deterministic, which means that from a given state and input symbol, the next state will *always* be the same. Note that in order to get to a state V_i or W_j, only one set of input symbols is allowed. For example, the state V_1 can only be reached through the input symbols $d_1, \ldots, d_i$ from a subset of the W_js.

[1]Some like to define a *dump state* where all rejected strings finally end up. This dump state has not been included in the figure

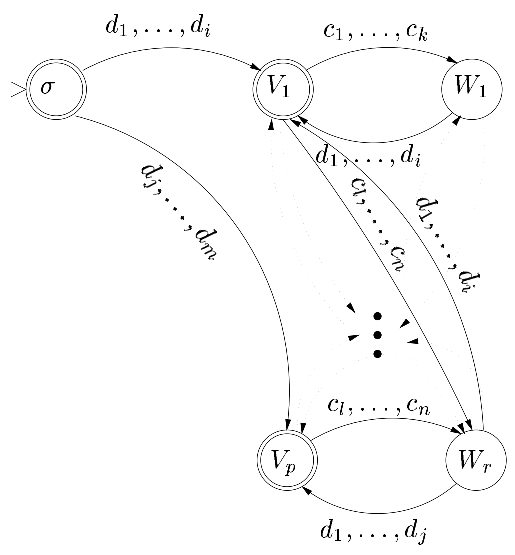

Fig. 3. Finite state automaton (FSA)

The automaton can also be used as a basis for a modelling assistant where a code generator checks model consistency and generates basic source code from the accepted strings. This source code can then be refined by the modeller before compilation.

Algorithm to generate strings from a directed graph

We have now looked at how grammars and automata can be applied in consistency checking and being a modelling assistant. We will now discuss the modelling from a slightly different angle.

Given a network as shown in figure 1 describing the process. The diagram is a directed graph, from which we may produce legal words (Gill 1976). By assigning a symbol to each node in the figure, it is possible to parse the graph in order to obtain the strings of a language describing the process.

The following algorithm can be used to generate the strings:

1. Start in an device node that has not yet been traversed.
2. Follow an unvisited link to a connection. If such a link does not exist, the device's symbol is a string in the language. Go to 7
3. Follow an unvisited link from this connection to a device.
4. The symbols of the three devices form a string in the language.
5. Repeat from 3 until all links are visited.
6. Repeat from 2 until all links are visited.
7. Repeat from 1 until all devices in the graph have been visited.

The set of strings produced by this algorithm contains strings with 1 or 3 symbols only. More sophisticated algorithms which will produce longer strings are possible. This one is chosen for convenience.

We see that the formal language description of a dynamic process model is equivalent to the component flow diagram in figure 1.

Deriving ODEs from the strings

We may produce a set of differential equations from each device (from the **ED**-set), based on the component network (elements from **A**, **T** and **R**) present inside the device, and connections to other devices (elements in the **EC**-set). An algorithm able to generate the differential equation structure has been implemented. The flow and reaction laws must be provided either by the modeller, or from a model base. Automatic generation of equations from a phenomenological process description will in general lead to a set of equations with index problems (Moe 1995). Assumptions and constraints regarding e.g. equilibrium reactions and constant volume and pressure will lead to such problems. These problems are also discussed in Drengstig *et al.* (1996), and are topic for current research.

2.5 Comparing and combining OO-model and FLD

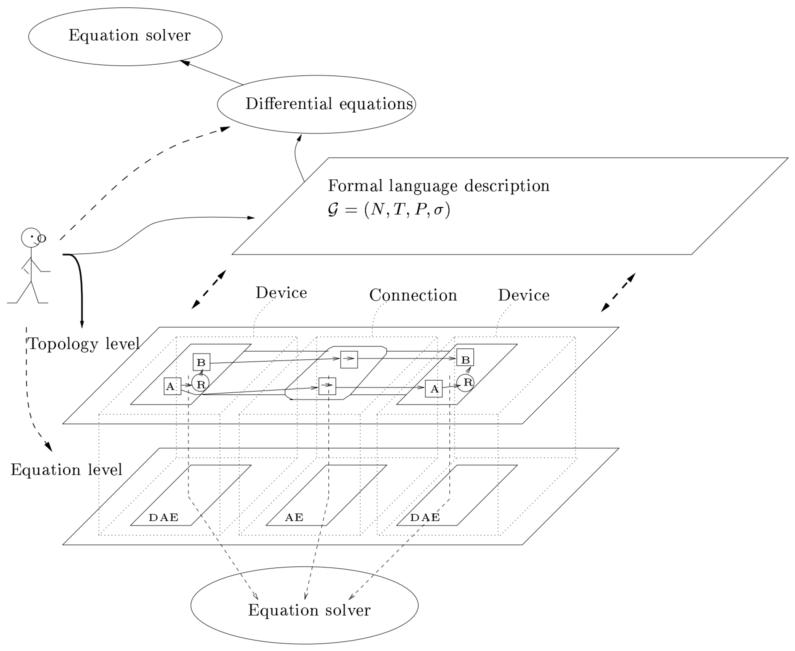

Fig. 4. Object-oriented model and formal language description

Figure 4 shows how the OO model is constructed. The model consists of a topology level, where the internal topology (component network) of each object is defined. In addition a set of equations define the behaviour of the object. The objects (represented by dotted boxes) are connected in the total model. The object equations are solved simultaneously in an external equation solver.

The alternative FLD is equivalent to the topology level of the OO model. Equation sets can be generated by parsing the strings of the FLD. These equations can be sent to an equation solver. The automata based on FLD can be used to generate ODEs or OO-code to be compilied into a final model and simulator. The automata can also perform consistency checks on the proposed model, to make sure that the objects in the OO- description have the correct interfaces.

2.6 Discussion

Evaluating our concept, it is best suited to model unit processes where the model may be represented by explicit ODEs. Examples of processes where the modelling method may be applicable are:

- separation, e.g. distillation, absorption, liquid-liquid extraction, evaporation, separation by settling, surface properties or leaching.
- storage, i.e. buffer and storage tanks.
- chemical reactions, i.e. reactor tanks.

The reason for this is that the building blocks focus on volume decomposition, chemical components and energy, and mass and energy transfer. This implies that the potential for improving modelling efficiency is greatest for these types of processes.

The described two level decomposition of the process into elementary building blocks is one attempt to make process modelling more efficient. It is however not the only viable approach. In the following a few alternative methods will be discussed.

Equation based decomposition uses the different types of terms in an equation as building blocks. Powersim^{TM} (ModellData 1994) uses this decomposition strategy. The simulation tool provides a graphical interface which makes the model implementation very fast and flexible. It can be applied to different systems in a number of disciplines (process modelling, economy, etc.). A disadvantage is that the model decomposition does not reflect the topology of the process, nor does it show explicitly the phenomena taking place. In addition, the encapsulation principle is poorly supported. This may cause some maintenance difficulties if the model is large and needs revisions.

Our approach has many similarities with the phenomenon based decomposition used by Woods (1993). Each phenomenon as a building block rather than the terms in an equation. In addition, there is a description of the process unit's topology. The equations are generated based on the topology and the phenomena associated with each unit. The generated set of equations can then be simulated. The phenomena and the building blocks are explicitly defined, and should be easy to maintain.

3 EXAMPLE

We will investigate our method by the use of a semi-realistic example; a gas-filled tubular reactor. In this example we only look at one type of unit process, hence $\mathcal{M}$ will be small.

The reactor is shown in Figure 5. The gas-filled tubular reactor is connected to surrounding process units (called terminals in the figure). The reactor is filled with a gas consisting of 3 species: A, B and C. The exothermic reaction $A \rightarrow B + C$ takes place in the reactor. The reactor is air cooled. The air temperature is assumed to be constant. The reactor is described by partial differential equations. We assume, for convenience, that a model description where the reactor is divided into two control volumes is sufficiently good.

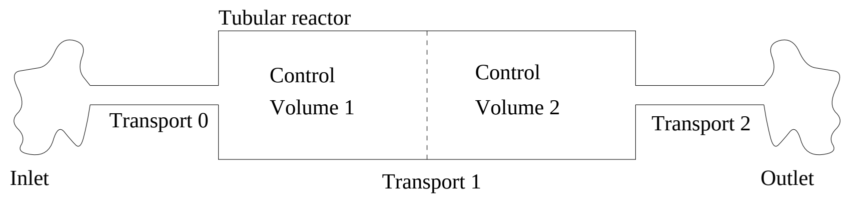

Fig. 5. Gas-filled tubular reactor, modelled as two control volumes and transport mechanisms.

3.1 The building blocks

From the process description above, we define building blocks to describe the modelled topology of the reactor; *control volume* (D_1) and *gas transport* (C_1). The building blocks are sufficient to describe the mass aspects and the convective part of the energy. The cooling of the reactor (conduction/radiation) requires an additional building block; *heat bridge* (C_2). The heat bridge contains information about the heat conduction coefficient from inside the reactor to the surroundings. In addition, terminating objects at the ends are needed. We create a building block called *terminal* (D_2).

Inside the *control volume*, there are chemical species and energy, and one reaction may take place. This means that the **A**-set is given by $\mathbf{A} = \{n_A, n_B, n_C, u\}$. There is one reaction, which means that $\mathbf{R} = \{R\}$. The flow of material is assumed to be given by convection of mass, while there are both heat conduction and heat convection in the process. Hence the **T**-set is given by $\mathbf{T} = \{F_{conv}, Q_{cond}, Q_{conv}\}$.

3.2 The process model

The model consists of model elements instantiated from the building blocks defined above, as shown in Figure 6. Instantiations of elements from **A**, **T** and **R** form the internal network within the devices and connections. The model of the unit process, i.e. the gas-filled tubular reactor, is given by the network of instantiated topological elements.

When the model has been constructed, it is given parameters and linked to a suitable integration method for simulation. Each of the **ED**-objects calculate a vector of derivatives ($\underline{\dot{x}} = \underline{f}(\cdot)$) based on the contents of the **ED**. We use an explicit integration routine which returns a vector of updated states.

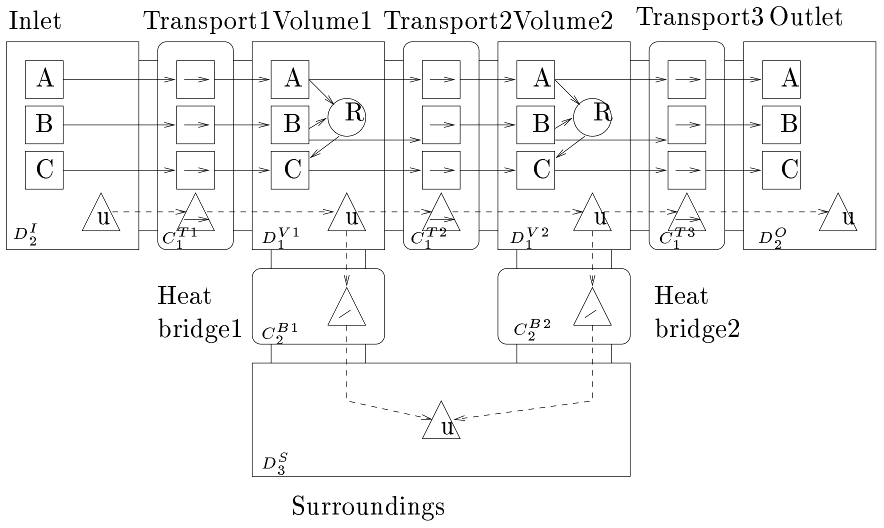

Fig. 6. Reactor model. Species and reactions (elements of **A**, **T** and **R**) define internal networks inside the **EC**- and **ED**-elements. Square phenomenological symbols represent mass aspects, while triangles represent energy. Round symbols represent reactions.

3.3 Process model description using a formal language

The set of entities and phenomena in this example is given above. By convention, we define upper case symbols to represent mass aspects, while lower case symbols represent energy aspects.

The topological level building blocks are given by: $\mathbf{EC} = \{C_1, C_2\}$, and $\mathbf{ED} = \{D_1, D_2, D_3\}$. The sets of instantiated symbols that belong to the elements are given by:

$$\Sigma_{C_1} = \{t_A, t_B, t_C, t_u\} \tag{12}$$
$$\Sigma_{C_2} = \{t_u\} \tag{13}$$
$$\Sigma_{D_1} = \{a_A, a_B, a_C, r, a_u\} \tag{14}$$
$$\Sigma_{D_2} = \{a_A, a_B, a_C, a_u\} \tag{15}$$
$$\Sigma_{D_3} = \{a_u\} \tag{16}$$

The Σ-sets forms the *alphabet*s used to describe the legal strings on the phenomenological level. The strings are formed (or checked for consistency), using the grammar $\mathcal{G}$, using the alphabets as sets of terminals, T. The set of strings describing the system is denoted M. This is a subset of the language, $L(\mathcal{G})$.

The following set of strings describe the topological level before the devices and connections are linked:

$$M_{C_1} = \{t_A, t_B, t_C, t_u\} \tag{17}$$

$$M_{C_2} = \{t_u\} \tag{18}$$

$$M_{D_1} = \{a_A r a_B, a_A r a_C\} \tag{19}$$

$$M_{D_2} = \{a_A, a_B, a_C, a_u\} \tag{20}$$

$$M_{D_3} = \{a_u\} \tag{21}$$

Since the only phenomenon present in the building blocks is the reaction r in the control volume D_1, the set M_{D_1} is the only one with strings longer than one symbol. The string $a_A r a_B$ means that the component A is transformed to B by the reaction r. A is also transformed into C by the same reaction.

From Condition 1 we find that the allowable connections between the topological elements are: C_1^i may connect D_1^j and D_2^k, since Σ_{C_1} is a subset of both Σ_{D_1} and Σ_{D_2}. C_2^i may connect all **ED**-elements, since Σ_{C_1} is a subset of all the alphabets of the **ED**-elements (i.e. $\Sigma_{D_1}, \Sigma_{D_2}, \Sigma_{D_3}$).

We now define an alphabet consisting of symbols from the topological level to construct the topological level model. The process has two terminals: D_2^I (inlet) and D_2^O (outlet), both of type D_2. The surroundings D_3^S is of type D_3. There are two control volumes D_1^{V1} and D_1^{V2}, three transports C_1^{T1},C_1^{T2},C_1^{T3}, and finally the heat bridges C_2^{B1} and C_2^{B2}.

These elements form the alphabet Σ_M on the topological level. The set of terminating symbols in the grammar $\mathcal{G}_M$ is given by: $T_M = \Sigma_M = \{C_1^{T1}, C_1^{T2}, C_1^{T3}, C_2^{B1}, C_2^{B2}, D_1^{V1}, D_1^{V2}, D_2^I, D_2^O, D_3^S\}$.

The total model is described by:

$$\begin{aligned} M_M = \{ & D_2^I C_1^{T1} D_1^{V1} C_1^{T2} D_1^{V2} C_1^{T3} D_2^O, \\ & D_1^{V1} C_2^{B1} D_3^S, D_1^{V2} C_2^{B2} D_3^S\} \end{aligned} \tag{22}$$

This set of strings describe the model on the topological level, and $M_M \subset L(\mathcal{G}_M)$. The set of strings is not unique. Other possible set would be: $M_M^{'} = \{D_2^I C_1^{T1} D_1^{V1}, D_1^{V1} C_1^{T2} D_1^{V2}, D_1^{V2} C_1^{T3} D_2^O, D_1^{V1} C_2^{B1} D_3^S, D_1^{V2} C_2^{B2} D_3^S\}$, or $M_M^{''} = \{D_2^I C_1^{T1} D_1^{V1} C_2^{B1} D_3^S, D_1^{V1} C_1^{T2} D_1^{V2} C_2^{B2} D_3^S, D_1^{V2} C_1^{T3} D_2^O\}$.

We may run a consistency check (using for instance an automaton) on the sets to ensure that the strings are correct, and, hence, that the model building blocks are correctly assembled. We may also produce a set of differential equations of the model.

3.4 Retrieving equations from strings or graphs

The basic differential equations can be retrieved from the strings or from a graph similar to the one in figure 6. Since the set of strings and the graph are equivalent representations, we will demonstrate how the differential equations can be retrieved from the strings. The algorithm is defined as follows:

- The differential variables are defined by the elements of Σ_{D_i}.
- External flux terms are added according to the links determined by the transports to the neighbouring connections.
- Internal flux terms are added according to the reactions.

The following ODEs were produced:

$$\dot{n}_{A,D_2^I} = -F_{A,C_1^{T1}} \tag{23}$$

$$\dot{n}_{B,D_2^I} = -F_{B,C_1^{T1}} \tag{24}$$

$$\dot{n}_{C,D_2^I} = -F_{C,C_1^{T1}} \tag{25}$$

$$\dot{u}_{D_2^I} = -Q_{u,C_1^{T1}} \tag{26}$$

$$\dot{n}_{A,D_1^{V1}} = -R_{D_1^{V1}} + F_{A,C_1^{T1}} - F_{A,C_1^{T2}} \tag{27}$$

$$\dot{n}_{B,D_1^{V1}} = R_{D_1^{V1}} + F_{B,C_1^{T1}} - F_{B,C_1^{T2}} \tag{28}$$

$$\dot{n}_{C,D_1^{V1}} = R_{D_1^{V1}} + F_{C,C_1^{T1}} - F_{C,C_1^{T2}} \tag{29}$$

$$\dot{u}_{D_1^{V1}} = Q_{u,C_1^{T1}} - Q_{u,C_1^{T2}} - Q_{u,C_2^{B1}} \tag{30}$$

$$\dot{n}_{A,D_1^{V2}} = -R_{D_1^{V2}} + F_{A,C_1^{T2}} - F_{A,C_1^{T3}} \tag{31}$$

$$\dot{n}_{B,D_1^{V2}} = R_{D_1^{V2}} + F_{B,C_1^{T2}} - F_{B,C_1^{T3}} \tag{32}$$

$$\dot{n}_{C,D_1^{V2}} = R_{D_1^{V2}} + F_{C,C_1^{T2}} - F_{C,C_1^{T3}} \tag{33}$$

$$\dot{u}_{D_1^{V2}} = Q_{u,C_1^{T2}} - Q_{u,C_1^{T3}} - Q_{u,C_2^{B2}} \tag{34}$$

$$\dot{n}_{A,D_2^{O}} = F_{A,C_1^{T3}} \tag{35}$$

$$\dot{n}_{B,D_2^{O}} = F_{B,C_1^{T3}} \tag{36}$$

$$\dot{n}_{C,D_2^{O}} = F_{C,C_1^{T3}} \tag{37}$$

$$\dot{u}_{D_2^{O}} = Q_{u,C_1^{T3}} \tag{38}$$

$$\dot{u}_{S_0} = Q_{u,C_2^{B1}} + Q_{u,C_2^{B2}} \tag{39}$$

F_X in these equations refers to molar flow between devices, R_X refers to internal molar flow because of reactions. Q_X refers to heat flow between devices. The most interesting part here is the energy balance. There are three important assumptions that must be fulfilled for the energy balance to be correct:

1. Note that the reaction heat is not present in the equations. This is correct only if the heat of formation of the species is included in the enthalpy for the flowing material.

2. It is assumed that the internal energy is the dominant part of the energy term. This is a common assumption in a large class of modelling problems.

3. Shaft work is neglected.

Note that the equations may be dependent. Some of the equations may not be necessary and can be removed. Redundant differential equations may be removed, and replaced by algebraic equations. In some cases these redundancies can be identified directly from equation sets. Often, however, such redundancies cannot be resolved until a full, detailed model, where the equations defining all the flows, etc. are available.

Equations 23-26 and 35-38 will typically be deleted, since the terminals usually contain boundary conditions that should not be changed. The algorithm presents the largest possible set of ODE-candidates, based on the model representation of the input string. It is left to the modeller to determine which equations should be removed.

The above example is a very simple one, without any feedback loops. Such loops will, however, not affect the model representation. The algorithm performing the consistency check must have mechanisms to avoid infinite looping. This is also the case for the ODE-producing algorithm.

One of the interesting points here is that relatively large sets of ODEs can be represented by a small set of strings in a FLD, i.e. equations 17-21 and 22.

The methodology presented shows that it is possible to generate the differential equation structure for a modelling problem automatically, from a phenomenological description. As the details in the modelling problem are revealed (e.g. reaction rates, flow rates), empirical mathematical relations are used to describe the behaviour of the system, and the topological and phenomenological description is less useful. Therefore, at some point of detail, a mathematical description must be included.

4 CONCLUSION

We have looked into some of the problems related to automatic generation of differential equations based on a topological and phenomenological description of a process. The relation between model topology, differential equations and formal language representation has been demonstrated. Formal language representation can be used as a compact representation of a set of ODEs, and can also be used to check consistency in object-oriented models.

NOMENCLATURE

This section contains a list of the symbols used in this paper, and a description of how they should be interpreted. Some symbols are omitted. This is the case for symbols that are only used in a definition, and have little relevance for the further understanding of the contents.

EC	Elementary connections. A set of topological level building blocks which model *transports* of some kind.
ED	Elementary devices. A set of topological level building blocks which model *accumulations* of some kind.
A	Accumulation. A set of phenomenological level building blocks. The elements in the set have the ability to accumulate inside **ED** elements.
T	Transport. A set of phenomenological level building blocks which describe transport between elements of the **A**-set.
R	Reaction. A set of phenomenological level building blocks which describe reactions between elements of the **A**-set.
$\mathcal{M}$	Set of models.
Σ	An alphabet.
Σ_X	Finite alphabet used in the context given by X. X may be a building block or a model.
$\mathcal{G}$	A grammar.
$\mathcal{G}_X$	A grammar used in the context given by X. X may be a building block or a model.
N	Nonterminals in a grammar. N in a regular grammar corresponds to the set of states S in an automaton.
T	Terminals in a grammar. T in a regular grammar corresponds to the set of input symbols X in an automaton.
M_X	is the set of strings describing the topology of the components within X. X may be a building block or a model.
$L(\mathcal{G}_X)$	A language, i.e. all allowable strings that can be produced by the grammar $\mathcal{G}_X$.

FORMAL LANGUAGE DEFINITIONS

The following definitions related to formal languages are taken from Ginsburg (1975) and Gill (1976).

Definition 2
- An alphabet Σ is a set of abstract symbols.
- A string over an alphabet Σ is a finite sequence of symbols in Σ.
- Let the string of length zero, called the *empty* string, be denoted by λ.
- The set of all strings, over an alphabet Σ is denoted by Σ^+. Let $\Sigma^* = \Sigma^+ \cup \lambda$.
- For a finite alphabet Σ, each subset $L \subseteq \Sigma^*$ is called a *language*.

Definition 3
- A phrase-structure grammar is a 4-tuple:

$$\mathcal{G} = (N, T, P, \sigma) \tag{40}$$

  where

N	is a finite nonempty set of nonterminals,
T	is a finite nonempty set of terminals ($T \cap N = \emptyset$),
P	is a finite nonempty set of productions, and
σ	is the start symbol, $\sigma \in N$.

- A language generated by a phrase-structure grammar is called a phrase-structure language, and is denoted $L(\mathcal{G})$.
- Context-free grammars are those in which all production rules are of the form $A \rightarrow \beta$, where $A \in N$, $\beta \in (N \cup T)^+$. Languages generated by such grammars are called context-free languages.
- Regular grammars are those in which all production rules are of the form $A \rightarrow aB$ or $A \rightarrow a$, where $A, B \in N$, $a \in T$. Languages generated by such grammars are called regular languages.

The term *formal language* is often used to distinguish this type of language from *normal* languages.

Definition 4 A finite-state automaton is given by:
$\mathcal{A} =< S, X, f, \sigma_0, F >$ where

- S is a finite nonempty set called the state set, consisting of states.
- X is a finite nonempty set called the input alphabet.
- f is a function $f : S \times X \rightarrow S$ called the *next-state function*
- $\sigma_0 \in S$ is the initial state
- $F \subset S$ is a set of accepting states

From the regular grammar above, it is fairly easy to recognise the sets:

- $S = N = \{\sigma, V_1, V_2, \ldots, V_r, W_1, W_2, \ldots, W_p\}$
- $X = T = \{c_1^1, c_1^2, \ldots, c_1^s, c_2^1, c_2^2, \ldots, c_m^t, d_1^1, d_1^2, \ldots, d_n^u, \lambda\}$
- $\sigma_0 = \sigma$
- $F = \{\sigma, V_1, V_2, \ldots, V_r\}$

GRAPHICAL SYMBOLS USED

This section presents the graphical symbols used in the graphs throughout this article.

Elementary device	□	
Elementary connection	▯	

Table 1. Topological symbols

mass accumulation energy accumulation	□ △	
reaction	○	
mass convection mass diffusion	⍈ ⍂	
heat convection heat conduction radiation	△→ △\\ △✳	

Table 2. Topological symbols

Acknowledgement

This paper has been financially supported by The Norwegian Research Council and Elkem a/s. Dr.Ing. Ommund Øgård at SINTEF

Automatic Control has contributed with helpful hints and suggestions. We acknowledge the work done by Roy Ove Lie in his Master's thesis, which has inspired some of the work presented in this paper.

REFERENCES

Drengstig, T., S. O. Wasbø and B. A. Foss (1996). A formal graphical based process modeling methodology. Technical report. Department of Engineering Cybernetics, Norwegian University of Science and Technology, Trondheim, Norway.

Eikaas, T. I. (1990). Cadas - a system for computer aided design, analysis and synthesis of industrial processes. In: *Nordic CACE Symposium, Lyngby, Denmark.*

Elmqvist, H. (1978). A Structured Model Language for Large Continuous Systems. PhD thesis. Lund Institute of Technology.

Gill, A. (1976). *Applied algebra for the computer sciences.* Prentice-Hall.

Ginsburg, S. (1975). *Algebraic and automata-theoretic properties of formal languages.* North-Holland/American Elsevier.

Marquardt, W. (1994). Trends in Computer-Aided Process Modeling. In: *Proceedings of PSE'94.* pp. 1–24.

Mattson, Sven Erik and Mats Andersson (1992). Omola - an object-oriented modeling language. In: *Recent Advances in Computer-Aided Control Systems Engineering* (M. Jamshidi and C. J. Herget, Eds.). Elsevier Science Publishers B.V.

Mjaavatten, A. (1994). *Topology Based Diagnosis for Chemical Process Plants.* Dr. Ing. thesis. Department of Engineering Cybernetics, Norwegian Institute of Technology, Trondheim.

ModellData (1994). *Powersim 2.0, User's Guide and Reference.* ModellData.

Moe, H. I. (1995). *Dynamic Process Simulation. Studies on Modeling and Index Reduction.* Dr. Ing. thesis. Department of Chemical Engineering, Norwegian Institute of Technology, Trondheim, Norway.

Nilsson, B. (1993). Object-Oriented Modeling of Chemical Processes. PhD thesis. Lund Institute of Technology.

Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy and W. Lorensen (1991). *Object-Oriented Modeling and Design.* Prentice Hall.

Singstad, P. (1992). *Modelling and Multivariable Control of High Pressure Autoclave Reactor for Polymerization Reactor.* Dr. Ing. thesis, Department of Engineering Cybernetics, Norwegian Institute of Technology, Trondheim, Norway.

Wasbø, S. O. (1996). *Ferromanganese Furnace Modelling Using Object-Oriented Principles.* Dr. Ing. thesis. Department of Engineering Cybernetics, Norwegian University of Science and Technology, Trondheim, Norway.

Wasbø, S. O. and B. A. Foss (1995). An Object-Oriented Method for Process Modelling. In: *Preprints of DYCORD+'95* (J.B. Rawlings, Ed.). Danish Automation Society. pp. 117–122.

Winblad, A. L., S. D. Edwards and D. R. King (1990). *Object-Oriented Software.* Addison-Wesley.

Woods, E. A. (1993). *The Hybrid Phenomena Theory.* Dr. Ing. thesis. Department of Engineering Cybernetics, Norwegian Institute of Technology, Trondheim, Norway. ISBN 82-7119-520-4.