Chapter 14

Metamodeling of Hydrogen Supply Chains: A Programmable Structure Based Representation

Monika Varga and Bela Csukas
Research Group on Process Network Engineering, Institute of Methodology, Kaposvar University, Kaposvar, Hungary

14.1 INTRODUCTION

14.1.1 Basic Approaches for Modeling of Hydrogen Supply Chains

The need for the energy transition is an accepted fact worldwide. According to the concise summary of the Hydrogen Council (2017), "hydrogen can be used as fuel for power or in industry as feedstock, it can be produced from (renewable) electricity and from carbon-abated fossil fuels, it produces zero emissions at point of use, it can be stored and transported at high energy density in liquid or gaseous form, and it can be combusted or used in fuel cells to generate heat and electricity." In transforming energy systems, hydrogen can serve as a clean and safe energy carrier to pave the way toward a low-carbon economy.

Considering the slow but definite development toward a hydrogen economy, the optimal design, planning, and operation of a hydrogen supply chain (HSC) has come to the forefront of research interest in the last decade. Several design and planning case studies have been carried out in the past years. This Chapter does not intend to give a detailed review, because many really comprehensive review papers are already available on this topic. This introduction refers to some of them in the context of the challenges for model-based analysis and problem solving.

Dagdougui (2012) reviews the state of the art of the generally applied modeling approaches in the planning and design of HSC with almost 100 references. The review distinguishes three main methodological approaches, namely the mathematical optimization methods, the spatial (GIS based) models and frameworks, as well as the assessment plans/transition scenarios. On the basis of the comprehensive review, the author concludes that mathematical

Hydrogen Supply Chain. https://doi.org/10.1016/B978-0-12-811197-0.00014-2

programming–based optimization methods are widely used for the design and planning of future HSCs.

Five years later, in a recent paper Maryam (2017), reviews again the applied modeling approaches. From the almost 100 references overviewed, around 50% published in the period of 2013–17. Focusing on the context of the United Kingdom, similar methodologies (various mathematical optimization methods, such as MILP, MINLP, MPP, MOP; GIS based approaches; transition models) support the tracking of progress during the last 5 years. The review confirms that the mathematical programming–based optimization methods are still the most widely used tools in planning and design, besides the GIS-based tools. The author also refers to some alternatively applied System's Dynamic and Agent Based approaches in the overview.

As a general review (not only HSC focused) of supply chain network design, a recent paper of Govindan et al. (2017) can be mentioned. It gives a comprehensive overview of more than 260 actual research papers and case studies for supply chain network design in various fields. As uncertainty is a critical factor of viable design decisions, the authors put the emphasis on the consideration of uncertainty in the model and in the optimization procedure. Uncertainty, both on the supply and demand sides, seems to be the most significant challenge also in the design of HSCs.

Turning to other challenges in HSC optimization, managing the multiple spatial and temporal scales can be mentioned. As Samsatli et al. (2016) highlight, the need for the combined consideration of various temporal scales also presents a notable challenge. While the representation of, for example, energy storage requires a short time scale (e.g., hourly), the consideration of a long planning horizon of many years may result in computationally intractable or very computationally intensive models.

Consideration of multiple criteria (e.g., costs, environmental sustainability, safety, etc.) is an obvious expectation during "green" supply chain design, but the incorporation of these issues generates another interesting challenge to be solved (Nurjanni et al., 2017).

14.1.2 Typical Elements of Hydrogen Supply Chains

Having studied some comprehensive case studies (De-Leon Almaraz, 2014; De-Leon Almaraz and Azzaro-Pantel, 2017), the typical elements of Hydrogen Supply Chains are allocated at geographically determined sites or districts that can be identified by an appropriate grid or by specific districts in a GIS. Many elements are time-specifically controlled by the meteorological or seasonal characteristics. The typical elements can be classified, as follows.

- Primary renewable resources,
- Primary accumulated resources,
- Processing of primary resources,

- Intermediate energy containing products and byproducts,
- Intermediate transportations and transformations of the energy containing products and byproducts,
- Hydrogen production sources,
- Hydrogen production processes,
- Produced raw hydrogen,
- Conditioning processes of the produced hydrogen,
- Production site storage of hydrogen,
- Transportation of hydrogen,
- Consumption site storage of hydrogen,
- Utilization of hydrogen,
- Hydrogen-based produced energy and materials,
- Car fuel demand,
- Other pools to be taken into consideration.

Primary renewable resources of hydrogen production are solar radiation, the wind, and hydropower generated by the environmental system. The utilization of these resources (especially of the solar energy) needs *natural or cultivated land* that is a special finite resource shared with other uses (agriculture for food, etc.). Renewable resources and land are distributed in the districts of the investigated territory.

The *primary nonrenewable, accumulated resources* are the nuclear raw materials, the natural gas, and the coal. Nuclear raw materials need vehicles to transport them to the specifically located power stations, while natural gas is available from pipelines. In contrast, coal-based processing should be installed near the mining sites.

Processing of the primary resources can be solved by many well-developed and promising new methods, but all of them depend on hourly and seasonally changing environmental conditions (e.g., sunshine). Direct *utilization of the solar energy* is solved by various photovoltaic processes with a limited efficiency and capacity. There is intensive research and development for many new solutions, with special organisms (e.g., microalgae) or with artificial biosystems (e.g., artificial chloroplast). Some of them might cause revolutionary change; accordingly, the long-term economic evaluation that is currently applied may lead to incorrect decisions. The indirect utilization of solar energy is realized by cultivation of energy plants, which competes with land use for food production. The products appear seasonally.

Wind power stations and some smaller hydropower stations produce electrical energy in certain (often randomly changing) periods, so their effective use needs onsite conversion to electrical energy. Large hydropower and nuclear stations produce easily transportable electrical energy that can contribute to buffered hydrogen production by remote electrolysis.

The most frequently applied Steam Methane Reforming (SMR) competes with other uses of natural gas but can produce hydrogen-containing gas

anywhere in the vicinity of pipelines. In contrast, coal gasification has to be conducted near the resource, while it produces more expensive hydrogen-containing gas from a resource that has less convenient traditional applications.

There are *intermediate energy containing products and byproducts* in HSC networks, which originate from many distributed districts. Energy producing biomass (energy plants), as well as some agricultural byproducts and waste appear seasonally, however municipal biological wastes are formed permanently.

These intermediate energy-containing materials usually need *transportation, followed by transformation* into hydrogen-containing gas. Biomasses are processed by gasification, while biological wastes produce biogas by anaerobic fermentation. There are some experimental results on how to enrich the hydrogen content in the fermentation process (e.g., Wang and Yin, 2017).

The energy containing, *direct hydrogen producing sources* are the electrical energy, resulting from various chains, and the hydrogen-containing gas, coming from SMR and gasification processes. The two corresponding kinds of *hydrogen producing processes* are electrolysis and the different methods of gas separation for hydrogen enrichment (e.g., membrane separation, pressure swing adsorption, etc.).

The *produced raw hydrogen has to be conditioned* at the production site by liquefaction or by compression. The resulting liquid or compressed hydrogen product needs *production site storage* of liquid and gaseous hydrogen, respectively.

The *transportation of hydrogen* from the production to the consumption sites can be solved by tanker trucks, tube trailers, railway tankers, railway tube cars, or pipelines. Optionally the gases of high hydrogen content can be added to the available natural gas distribution pipelines.

Pure liquid or gaseous hydrogen requires *consumption site storage*. The *utilization of hydrogen* is connected to these storage places. In addition to the fueling of cars and other vehicles (e.g., aircraft), the hydrogen can be used for the generation of thermal energy, as well as for various industrial processes (e.g., ammonia production, synthesis of organic products, metal processing, etc.).

The logistics process is controlled by the *site and time dependent demands* for the fuels, heating, and other uses.

The evaluation of the various HSC process systems requires the calculation of the effects on balances of some *globally important pools,* such as carbon dioxide, oxygen, water, and thermal energy.

14.1.3 Challenges of Process Modeling for Hydrogen Supply Chain Design and Operation

In the literature, the basic approach to HSC design, planning, and operation is mathematical programming–based optimization. In these solutions, a collaboration with uncertainty model is embedded in the usually superstructure-based

multiobjective optimization program. In thinking about an alternative, unconventional approach, we try to summarize the major challenges to be addressed by the potential methodologies. Based on the literature and the above discussed characteristics of the typical elements in the Hydrogen Supply Chain, the challenges can be summarized, as follows:

- Considering the intensive scientific and technical development (especially the emerging new bio-based hydrogen production methods), modeling has to prepared for modifications involving optional new structures and process elements;
- Extensibility of the model is another important issue. Accordingly, the stepwise structural and functional changes of supplies and demands have to be taken into consideration without the reformulation of the model;
- Easy consideration of multiple sites and districts (compartments) is necessary (e.g., via optional connection of spatial coordinates of the elements with an external GIS tool);
- The methodology has to tolerate also case-specific multiscale extensions (i.e., part of the elements needs a more detailed model and/or finer spatial or temporal resolution);
- Part of the elements and structural connections need a time-driven execution at given points of time and/or during prescribed intervals;
- Material and energy balances have to be combined with signal and rule-based event-driven processes;
- The model and its elements must have built-in capabilities for robust communication with external (e.g., meteorological) databases and with collaborating (e.g., metaheuristic optimizing) programs;
- Considering the extraordinary uncertainty of the cost coefficients and some other parameters in a longer time-horizon, the model has to support the handling of the uncertain coefficients and data, and the respective collaboration with the external optimizer;
- The embedded elementary evaluations and evaluating elements have to be prepared for the multiobjective case, with a comprehensive set of time-invariant natural objective functions.

14.2 PROGRAMMABLE STRUCTURE BASED REPRESENTATION OF PROCESS SYSTEMS

The hybrid, multiscale models may be more complex than any specific mathematical apparatus. The presented *Programmable Structures* are motivated by the need that engineering analysis, design and control of the complex, multidisciplinary process systems have easily modifiable, extensible, and connectable methodologies, with common representation of functional and structural features.

14.2.1 Methodology of Direct Computer Mapping Based Programmable Structures

In *Direct Computer Mapping* (DCM) of process models (Csukas, 1998; Csukas et al., 2011) the natural building blocks of the elementary states, actions, and connections are mapped onto a unified set of building elements, which determine an executable program code, directly, without any specified mathematical apparatus. This principle was successfully applied for various processes from technological process systems (e.g., Csukas et al., 1999; Temesvári et al., 2004) up to the multisectoral agri-food processes (e.g., Varga et al., 2012).

Based on these previous applications and on the *recently developed multidisciplinary applications,* it is obvious that the structure of processes is more sophisticated than the networks in the sense of network science, on the one hand. However, the case-specific local functionalities of the elementary building blocks of these structures are less complicated than the systems of integral and partial differential equations, combined with discrete events, on the other hand. As an intermediate solution, we developed *Programmable Structures* (Varga et al., 2016a,b, 2017; Varga and Csukas, 2017) that can be generated from a network description and from two functional metaprototypes. The metaprototypes and the derived process structures are prepared for the semantically distinguished, but syntactically uniform representation of the "*model specific conservation law based, additive*" measures, and of the "*over-writable*" signals.

In this approach, the usual network or net-based formalization of process structures is extended to a special structure of unified state and transition elements, containing dedicated input and output slots for additive measures and signals, as well as parameter slots for parameters. The actual structure is defined by standardized connecting elements, corresponding to increases and decreases of (extensive) measures, as well as to reading and overwriting of the intensive properties and other qualitative or quantitative signals.

In various applications many state and transition elements can usually be modeled with the same local programs. Accordingly, we distinguish the actual elements and the program defining prototypes. The prototype elements contain symbolic input, parameter, and output variables, as well as local programs, using these variables and a limited number of global data. During the simulation, the actual elements start with the initial conditions and parameters, and the output values are recalculated stepwise with the knowledge of input and parameter data, according to the associated local program prototype. The distinguished input and output slots for extensive/intensive data and for signals support the combined execution of the balance-based and rule-based functionalities.

The data flow between the elements is determined by the connections. Both the actual elements, as well as the receiving and sending sides of the connections may be specified by a spatial identifier, determining the compartment and/or the level and/or the scale of the given entity. Also, both the elements

and the connections may have timing that defines temporal constraints for the execution, supporting the combined event-driven and time-driven execution of models.

Summarizing the methodology, the *programmable structure* of process models is generated from the description of the (optionally multilevel or multiscale) process network and from two functional metaprototypes. The case-specific, actual functional prototypes, containing the local models may be copied and edited from the general metaprototypes. The actual state and transition elements are parameterized and initialized concerning their case-specific prototypes. The actual model elements are executed by their associated prototypes. This execution and the connection-based communication between the state and transition elements of the *programmed structure* are solved by a general-purpose kernel program.

In the proposed model representation, the detailed description of possibility (design) space, as well as the measurements and the evaluating functions of the optionally multiple objectives, may be embedded in the state and transition elements. Accordingly, in identification and optimization tasks, the robust simulator may easily be combined with external metaheuristic optimizers.

The *major advantages* of the developed methodology come from the inherent and plausible coupling between the structural and functional knowledge, as well as from the extensibility and connectivity of the models.

In line with the well-known trade-off principle between being well-structured and effective, the *major disadvantage* of the methodology is less effectiveness compared with the case-specifically developed and implemented procedural methods and tools. These disadvantages might be compensated by the forthcoming micro-granularly parallel hardware/software implementations because the well-structured representation of local program prototypes supports these implementations.

14.2.2 Recent Implementation of Programmable Structures

The code of the recent implementation is written in declarative, logical Prolog language (GNU Prolog 1.4.4), which is a limited subset of first order predicate calculus. Prolog makes the easy description, and the unification-based execution, of the local program prototypes possible. This feature, combined with the applied, restricted, and standardized method of model description, supports the application of a general kernel for the quite different actual models. In addition, this declarative language offers a formal description thereby, which also fosters possible future implementations.

The automatic generation of the programmable structures starts from the GraphML-based graphical description of state and transition metaprototypes, as well as from the textual or graphical description of the process network. During generation, metaprototype elements are multiplied according to the description of the process network, supplemented with the connections between them.

It is to be noted, that the expert user can also start the model building with a GraphML editor from scratch. The graphical model can be edited further by various graph editors (actually yEd Graph Editor). In this step, the expert user makes copies from the metaprototypes and next declares the case-specific programs. This is followed by initialization and parameterization of the actual elements. Next, a Prolog program interprets the GraphML description into the facts and clauses, declaring the dynamic model. Finally, the general-purpose Prolog kernel program executes the various modes of dynamic simulation. This means a cyclically repeated processing of state elements, state → transition connections, transition elements and transition → state connections.

The *declarative model implementation* makes possible the application of list structures and recursive unification at any level of the description. Accordingly, the general description of the model is prepared for an optional number of list members, without the previous declaration of their length.

14.2.3 Previous and Ongoing Applications of Programmable Structures

Programmable Structures were successfully applied for quite different complex, multidisciplinary processes from low-scale cellular biosystems (Varga et al., 2017) through ecological (Varga and Csukas, 2017) and technological process systems (Csukas et al., 2013) up to Recirculating Aquaculture Systems (Varga et al., 2016a), and environmental process networks (Varga et al., 2016b). The advantage of this approach is that it gives flexibility for both structural and functional modification and extension of the process model.

The methodology supports the unified generation of reusable process models. The generated models are reconfigurable and extensible, and the models can be incremented from any "final" state. The robustness of dynamic simulation is based on the built-in control mechanisms, initiated by the automatic checking of feasibility bounds for the underlying "model specific conservation law based" measures.

The robust simulation fosters the dynamic model-based (sub)optimization with various, externally implemented metaheuristic methods (e.g., Genetic Algorithm). The basic principle, in accordance with the engineering way of thinking, is that instead of searching for an exact global optimum with a simplified model, it might be better to generate multiobjective suboptimal solutions from a more detailed model.

14.3 PROCESS NETWORK AND EXAMPLE FOR A SIMPLIFIED HYDROGEN SUPPLY CHAIN

14.3.1 Network and Net Representations of Process Systems

Starting from a simplified structural analysis of a process system, first we can recognize some relationships between the various state entities within a given

spatial compartment or between the various locations. These simplest structures can be represented by *networks*, defined as a relation in the set of state elements, as follows:

$$R \subset S \times S, \quad S_i \in S \tag{14.1}$$

where R = a relation, S = the set of state characteristics, and = S_i is the ith state property.

In graphical representation the elements of S correspond to the nodes, while R designates the edges between them. It is worth mentioning that the nowadays very popular "network science" and "network analysis" basically work with nondirected edges (as can be seen in Fig. 14.1A) that correspond to symmetric (compatibility) relations, while the Cartesian product according to Eq. (14.1) assumes ordered pairs (i.e., directed edges). Considering that the transformations, transportations, and rules in process systems are usually directed, in the process systems directed relations and edges are usually applied (see Fig. 14.1B), while the bidirectional changes have to be symbolized by two edges of opposite direction.

Moreover, in the process systems the transformations and rules can be represented by causally and/or stoichiometrically coupled edges (Varga et al., 2017). Accordingly, we must define these coordinated changes by introducing a second set (i.e., a second kind) of nodes, designating the transitions. The respective net structures were defined by (Petri, 1977), as follows:

$$N \subset (S \times T) \cup (T \times S), \quad S_i \in S, \; T_j \in T \tag{14.2}$$

where N = a net relation, S = the set of state elements, T = the set of transition elements, S_i = the ith state property, and T_j = the jth transition property. The well-known original Petri Nets (Petri, 1962) represent a special case of these nets.

In the graph representation of the net structure (Eq. 14.2), the state and transition nodes are distinguished, for example, by dots and bars, as illustrated in Fig. 14.1C. In these *process flux networks*, the transition nodes (signed with bars) correspond to the coordinated transportations, transformations, and rules, while the state nodes (signed with dots) represent the underlying state entities.

The net structure discussed above correctly represents the *flux structure of reaction or transportation* networks, in which the state → transition and transition → state edges of conservational processes follow the respective decreasing and increasing fluxes, but the causalities are not visible. However, in the *state/transition nets of signals and rules* the state → transition and transition → state edges follow the causalities, determined by antecedent and consequent signals.

Considering that in process modeling we use both additive quantities and signals, *in adequate process structures we must distinguish the additive measure decreasing and increasing, as well as the intensive property and other signal transferring connections*. In this representation of conservation law-based (conservational) processes (see Fig. 14.1D), the transitions read (dotted lines)

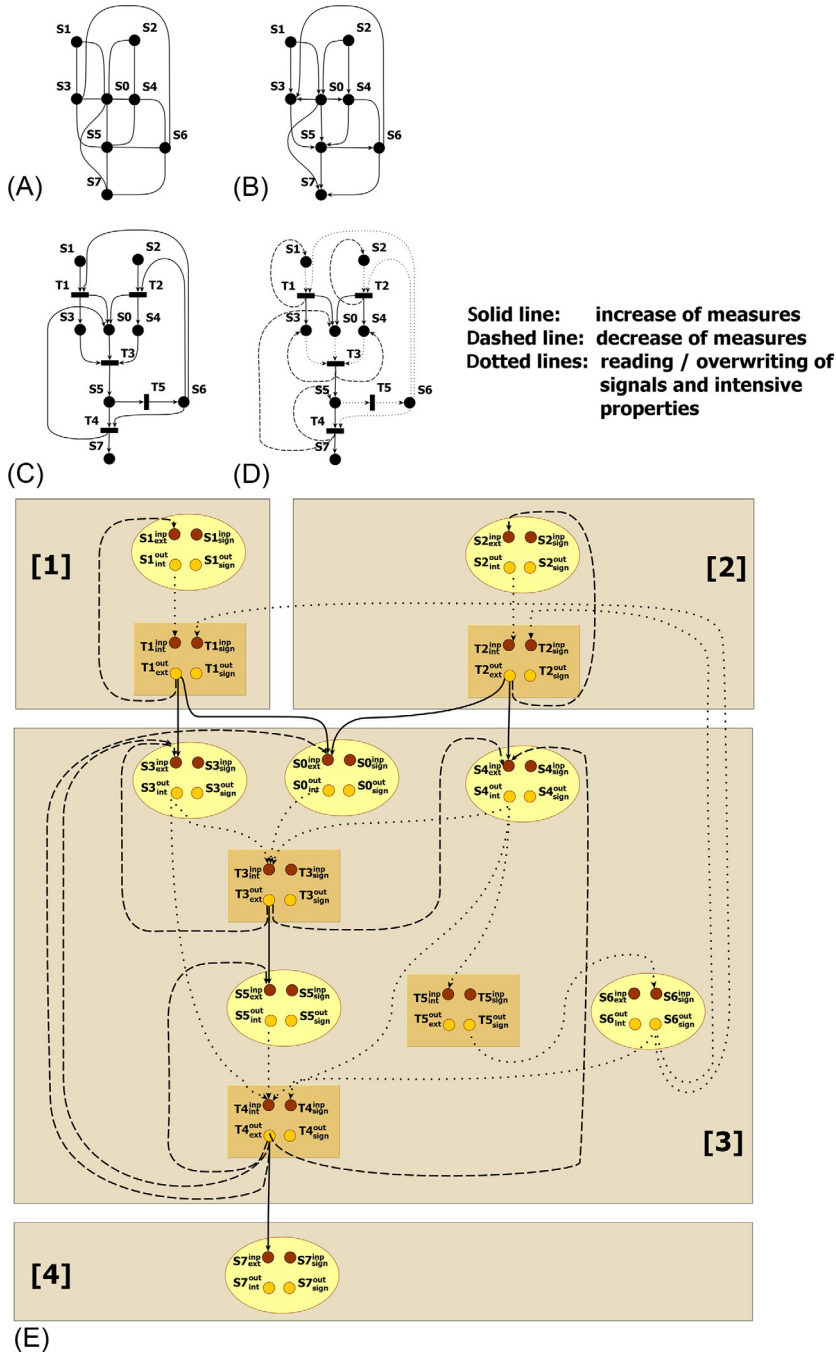


FIG. 14.1 Various structural representations of process systems. (A) Nondirected network, (B) directed network, (C) state-transition network/net, (D) process network/net, and (E) programmable structure representation of process network/net, involving compartments [1], [2], [3], and [4].

the rate determining intensive properties of the state elements, as well as increase (solid lines) and decrease (dashed lines) the extensive measures of the modified states. Similarly, in the rule-based signaling (informational) processes, the transitions read the antecedent signals and overwrite the consequent ones. It results in a uniform and causally right structural description for both conservational and informational processes (Varga et al., 2017; Varga and Csukas, 2017).

Accordingly, both state and transition elements must be *prepared for receiving input from, and sending output to, additive measures and signs*. These input and output connectors of the state and transition elements are called *slots* and are depicted by small circles, as shown in Fig. 14.1E. In this more sophisticated structure, the unified representation of the conservation law–based (quantitative) and rule-based (qualitative) edges, as well as the input/output characteristics of the nodes can be described uniformly and causally right. Accordingly, the above outlined *process nets* can be defined by the relation.

$$N \subset \left(S_{int}^{out} \times T_{int}^{inp}\right) \cup \left(S_{sign}^{out} \times T_{sign}^{inp}\right) \cup \left(T_{decr}^{out} \times S_{ext}^{inp}\right) \cup \left(T_{incr}^{out} \times S_{ext}^{inp}\right) \cup \left(T_{sign}^{out} \times S_{sign}^{inp}\right) \tag{14.3}$$

where in the indexes.

inp = input;
out = output;
int = intensive (derived from measures);
ext. = extensive (additive measure);
incr = increase (of an additive measure);
decr = decrease (of an additive measure),
sign = signal.

14.3.2 Illustration of a Simple Hydrogen Supply Chain Process Network

The process structure of a simple Hydrogen Supply Chain, in the sense of Eq. (14.3) and Fig. 14.1, is shown in Fig. 14.2. This is a small, enlarged part of the example structure, discussed in Section 14.4. The names, IDs and locations of the illustrated state and transition elements are summarized in Table 14.1. The districts, designated by 1, 2, 3, and 4 correspond to four fictitious, geographically outlined territories (e.g., according to a GIS shape file, as we combined it with a programmable structure of an environmental model (Varga et al., 2016b).

In the process structure, the state and transition elements are symbolized by ellipses and rectangles (Fig. 14.2), respectively. In both kind of elements, there are two input slots (signified with small red upper dots within the elements in

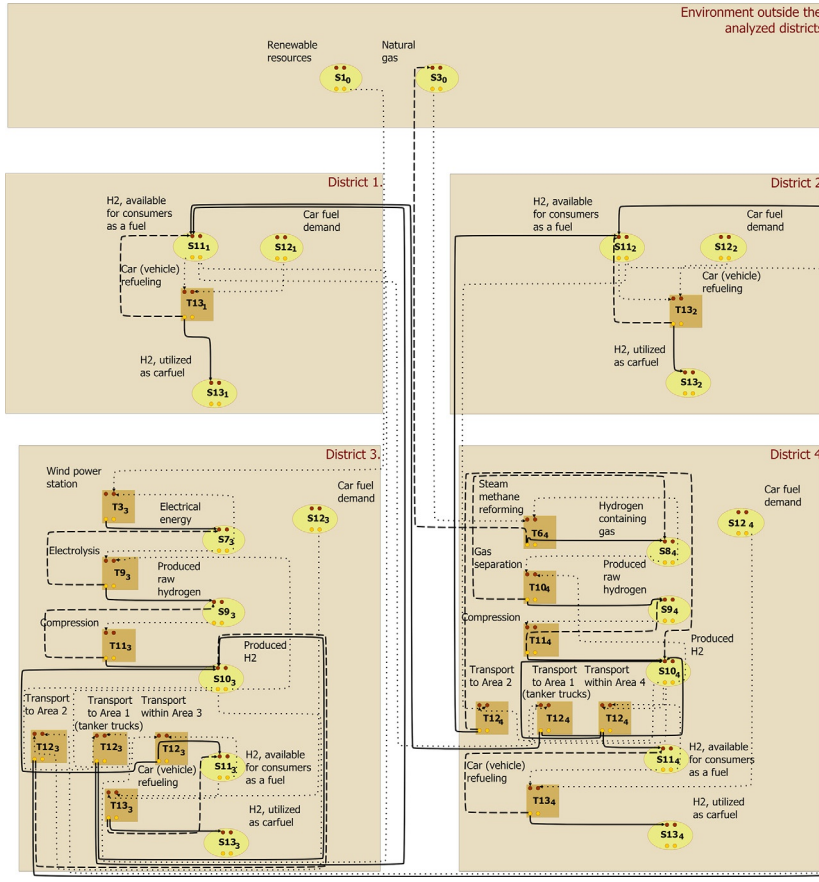


FIG. 14.2 Illustration of the process structure of a simplified part of a Hydrogen Supply Chain.

Fig. 14.2) for the input additive measures and over-writable signals, as well as two output slots (signified with small yellow lower dots within the elements in Fig. 14.2) for the output measure related properties and signals.

There are five kinds of connections, as follows:

- Increasing of (model-specific conservation law–based) measures: solid edges from measure change output slots of transitions to measure input slots of states;
- Decreasing of (model-specific conservation law–based) measures: dashed edges from measure change output slots of transitions to measure input slots of states;
- Reading/writing of measure-related (e.g., intensive) properties: dotted edges from measure-related output slots of states to the measure related input slots of transitions;

TABLE 14.1 Elements of the Simple Example Structure

	Name of Element	ID	District, Where the Given Material or Process is Involved
State elements	Renewable resources	S1	0
	Natural gas	S3	0
	Electrical energy	S7	3
	Hydrogen-containing gas	S8	4
	Produced raw hydrogen	S9	3, 4
	Produced compressed H2	S10	3, 4
	Compressed H2, available for consumers as a fuel	S11	1, 2, 3, 4
	Car fuel demand	S12	1, 2, 3, 4
	H2, utilized as car fuel	S13	1, 2, 3, 4
Transition elements	Wind power station	T3	3
	Steam methane reforming	T6	4
	Electrolysis	T9	3
	Gas separation	T10	4
	Compression	T11	3, 4
	Transport (tanker truck)	T12	3, 4
	Car (vehicle) refueling	T13	1, 2, 3, 4

- Reading/writing of transition output signals: dotted edges from signaling output slots of transitions to signaling input slots of states;
- Reading/writing of state output signals: dotted edges from signaling output slots of states to signaling input slots of transitions.

14.4 GENERATION OF THE PROGRAMMABLE STRUCTURE FOR A SIMPLE EXAMPLE HYDROGEN SUPPLY CHAIN

Hydrogen Supply Chains can usually be modeled by a large number of holistically connected state and transition elements, while their detailed

functionalities can be described by a significantly lower number of prototyped programs. Considering this, we start the model generation from the detailed graphical implementation of one state and one transition metaprototype, as well as from the description of the process network. Generation means multiplying the prototyped state and transition elements, as well as adding the connections between them, according to the actual structure, automatically. Actually, the graphically editable models can be generated from the GraphML definition of metaprototypes and from the textual description of the network structure, similarly to other process systems (Varga et al., 2017; Varga and Csukas, 2017).

In the following sections we shall use a limited Prolog syntax, in which:

upper case alphanumerical symbol = variable;
lower case alphanumerical symbol = constant;
Anything* = list of Anything;
[H|T] = list of any elements or functors with a head H and a list of tail T;
[] = empty list;
full stop = end of logical sentence;
if (or :-) implication in the sense of Horn clauses;
comma in the program body of clauses = and.

14.4.1 Declaration of the Metaprototypes

The structures of Hydrogen Supply Chains discussed above are built from state and transition elements. Also, there are two major classes of characteristics, namely the (often model-specific conservation law–based) extensive/intensive properties and the signals (signs). Accordingly, the primary metaprototypes, used for the generation of the actual structures should be prepared for the input and output of both extensive/intensive properties and signals.

The graphical representation of the general state and transition metaprototypes is shown in Fig. 14.3. In line with the representation of process models in Figs. 14.1E and 14.2, the state and transition prototypes are depicted by an ellipse and a rectangle, respectively. The outside communication and inside functionality related data structures, as well as the place for the local program codes are contained in the slots, as follows:

- balance measure and signal related input and data storage slots (small red dots),
- predefined general classes of local parameters (small red rectangles),
- place for the locally executable (actually declarative) program code of the various prototypes (small yellow diamonds), and
- balance measure and signal related data storage and output slots (small yellow dots),
- the reporting slot of the transitions that makes it possible to forward case specific data to the output stream of the simulation.

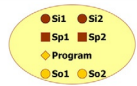
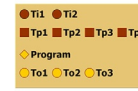
	State metaprototype	Transition metaprototype
		
Heading	{e(p,y,state,'Spatial',state,'Temporal','Possibilities','Evaluations')}	{e(a,y,trans,'Spatial',trans,'Temporal','Possibilities','Evaluations')}
Inputs	Si1 {i(comp,dl,'InpComps')} Si2 {i(inp,dl,'InpSigns')}	Ti1 {i(conc,dl,'InpConcs')} Ti2 {i(inp,dl,'InpSigns')}
Parameters	Sp1 {c(param,dl,'Parameters')} Sp2 {c(cond,dl,'Conditions')}	Tp1 {c(param,dl,'Parameters')} Tp2 {c(antecedent,dl,'Antecedent')} Tp3 {c(consequence,dl,'Consequence')} Tp4 {c(cond,dl,'Conditions')}
Outputs	So1 {o(conc,dl,'OutConcs')} So2 {o(out,dl,'OutSigns')}	To1 {o(comp,dl,'OutComps')} To2 {o(out,dl,'OutSigns')} To3 {o(report,dl,'Reports')}
Program	{program(' ')}	{program(' ')}

FIG. 14.3 Metaprototypes of state and transition elements.

The e(.) heading of the elements specifies:

- the type (p = passive state or a = active transition),
- the switchable existence (y = yes or n = no),
- the name of the metaprototype (for this application 'state' or 'trans'),
- the actual spatial scale (variable Spatial, unifying with a list of integers),
- the name of the prototype (in the metaprototype it replies the name of the metaprototype that has to be changed for the name of the derived prototypes in the course of editing the generated structure),
- the timing (variable Temporal),
- the design space (Possibilities, an empty list for the optional description of possibilities, associated with the given elements) and
- a list of data for the calculation of objectives (Evaluations).

The input, parameter, and output slots contain i(), c(), and o() functors. The i(), c(), and o() functors are described by a name, a data type (actually dl), and a list of the respective data. In case of data type dl, the data sets are described by lists of triplets.

```
d(Identifier, Variable_list, Instructions)
```

where.

Identifier = a symbol or a list, determining an identifying name;
Variable_list = a list of variables; and.
Instructions = instruction for the use (e.g., dimension, specification, etc.).

The program slot contains an empty place, starting with the *program ('* and ending with the *')* character sequences. In the course of the editing process, this place has to be filled with the declaration of case-specific prototypes.

The slots of the actually used *state metaprototype* (state, see Fig. 14.3 left) are the following:

- The balance measure input slot "comp" declares an empty or predefined list InpComps, for an optional number of d(.) triplets for the balance measure variables that summarize the increases and decreases, for the same d() triplet, as well as optionally new triplets that can be added or removed. The respective data come from the various transitions, via the connections, as can be seen in Fig. 14.2;
- The signaling input slot "inp" declares an empty or predefined list InpSigns, for an optional number of d(.) triplets for the sign variables that may be overwritten by new signs, as well as optionally new triplets that can be added or removed. The respective data come from the various transitions, via the connections, as can be seen in Fig. 14.2;
- The parameter slot "param" declares an empty list for the optional state parameters (described by d(.) triplets in the list Parameters);
- The condition slot "cond" declares an empty list for the optional conditions (described by d(.) triplets in the list Conditions);
- The measure-related output slot "conc" declares an empty list OutConcs, for an optional number of d(.) triplets for the calculated properties (e.g., available quantities, concentrations, temperature, etc.). The respective data are forwarded to the various transitions via the connections, as illustrated in Fig. 14.2. In addition, all of the output data can be optionally forwarded to the output recording;
- The signaling output slot "out" declares an empty list OutSigns for an optional number of d(.) triplets for the determined output signs. The respective data are forwarded to the various transitions via the connections, as illustrated in Fig. 14.2. In addition, all of the output data can optionally be forwarded to the output recording;
- The program code is to bind (to determine) all of the output variables (OutComps, OutSigns, Report) with the knowledge of the bound input (Spatial, Temporal, InpConcs, InpSigns) and parameter (Parameters, Antecedent, Consequence, Conditions) values. The expert-defined local prototype programs have to be prepared for the actual cases in edition of the generated programmable structure (see Section 14.5).

The slots of the actually used *transition metaprototype* (transition, see Fig. 14.3 right) are the following:

- The measure-related input slot "conc" declares an empty list InpConcs for an optional number of d(.) triplets for the variables, used for the calculation of the transitions. The respective data come from the various states, via the connections, as can be seen in Fig. 14.2;
- The signaling input slot "inp" declares an empty or predefined list InpSigns for an optional number of d(.) triplets for the sign variables, used for the

calculation of the transitions. The respective data come from the various transitions, via the connections, as can be seen in Fig. 14.2;
- The parameter slot "param" declares an empty list for the optional state parameters (described by d(.) triplets in the list Parameters);
- The antecedent slot "antecedent" declares an empty list for the optional antecedent signals (described by d(.) triplets in the list Antecedent);
- The consequence slot "consequence" declares an empty list for the consequence signals (described by d(.) triplets in the list Consequence);
- The condition slot "cond" declares an empty list for the optional conditions (described by d(.) triplets in the list Conditions);
- The measure change related output slot "comp" declares an empty list OutComps for an optional number of d(.) triplets for the calculated increases and decreases of measures. The respective data are forwarded to the various states via the connections, as illustrated in Fig. 14.2;
- The signaling output slot "out" declares an empty list OutSigns for an optional number of d(.) triplets for the determined output signs. The respective data are forwarded to the various states via the connections, as illustrated in Fig. 14.2;
- The reporting slot "report" declares an empty list Report for an optional number of d(.) triplets to forward case-specific data to the output stream of the simulation;
- The program code is to bind (to determine) all of the output variables (OutComps, OutSigns, Report) with the knowledge of the bound input (Spatial, Temporal, InpConcs, InpSigns) and parameter (Parameters, Antecedent, Consequence, Conditions) values. The expert-defined local prototype programs have to be prepared for the actual cases in edition of the generated programmable structure (see later on in Section 14.5).

The definition of the metaprototypes can be saved in GraphML files for further processing. We can also define other metaprototypes, however, the solution described above proved to be quite general in various applications (e.g., Varga et al., 2017).

14.4.2 Definition of an Example Network

The names, IDs, short symbolic names, and locations of the building elements of an illustrative, fictitious HSC example network are summarized in Table 14.2. The short symbolic names, used in the databases and programs, are written according to the Prolog syntax with lower case capitals. The districts, designated by 1, 2, 3, and 4 correspond to four fictitious, GIS represented territories.

The declaration of the process structure follows the definition Eq. (14.3), as well as the architecture of the metaprototypes discussed above. Accordingly, to

TABLE 14.2 State and Transition Elements of the Fictitious Example HSC

	Name of Element	ID	Short Symbol in the Model	District, Where the Given Material or Process is Involved
State elements	Renewable resources	S1	raw_renew	0
	Cultivated land	S2	raw_land	2, 4
	Natural gas	S3	raw_gas	0
	Nuclear energy raw materials	S4	raw_nucl	0
	Energy producing biomass	S5	biomass	2, 4
	Biological wastes	S6	biowaste	2, 4
	Electrical energy	S7	eleng	1, 2, 3, 4
	Hydrogen-containing gas	S8	h2contgas	2, 4
	Produced raw hydrogen	S9	hydrogen	1, 2, 3, 4
	Produced compressed H2	S10	h2prod	1, 2, 3, 4
	Compressed H2, available for consumers as a fuel	S11	h2cons	1, 2, 3, 4
	Car fuel demand	S12	demand	1, 2, 3, 4
	H2, utilized as car fuel	S13	h2carfuel	1, 2, 3, 4
	Heating energy	S14	h2heating	0
	Carbon-dioxide	S15	co2	0
	Oxygen	S16	o2	0

TABLE 14.2 State and Transition Elements of the Fictitious Example HSC —cont'd

	Name of Element	ID	Short Symbol in the Model	District, Where the Given Material or Process is Involved
Transition elements	Land cultivation	T1	t_cultivation	2, 4
	Photovoltaic process	T2	t_photovolt	1, 2, 3, 4
	Wind power station	T3	t_wind	2, 3
	Hydropower station	T4	t_hydro	3, 4
	Nuclear energy station	T5	t_nucl	1
	Steam methane reforming	T6	t_smr	2, 4
	Biomass gasification	T7	t_gasify	2, 4
	Anaerobic fermentation	T8	t_ferment	2, 4
	Electrolysis	T9	t_electrolysis	1, 2, 3, 4
	Gas separation	T10	t_gassep	2, 4
	Compression	T11	t_compression	1, 2, 3, 4
	Transport (tanker truck)	T12	t_tankertruck	1, 2, 3, 4
	Car (vehicle) refueling	T13	t_h2refuel	1, 2, 3, 4
	Heat production	T14	t_h2heat	1, 2, 3, 4

the applied *transition-based model description* (e.g., Varga and Csukas, 2017) the actual structure can be declared by the listing of state and transition elements, as well as by the declaration of the individual transitions by the state elements being connected to the different input and output slots of them. In the simplest models, all of the state and transition elements are in the same spatial

location (with the same Spatial identifier, say Spatial = []), accordingly the structure can be defined in Prolog syntax as follows:

```
states([],Statelist).
          Statelist = State*
transitions([],Transitionlist).
          Transitionlist = Transition*
trans(Transition,InpConcs,OutComps,InpSigns,OutSigns).
         InpConcs; InpSigns; OutComps; OutSigns = Part_of_Statelist
         Part_of_Statelist = State*
```

InpConcs, InpSigns, OutComps and OutSigns contain the names of states connected with measure input, signal input, measure output and signal output slots of transition.

In generation of multicompartment and/or multilevel and/or multiscale models, the states and transitions are declared with their respective Spatial coordinates, as follows:

```
states(Coord1,Statelist1).
states(Coord2,Statelist1).
...etc.
transitions(Coord1,Transitionlist1).
transitions(Coord2,Transitionlist2).
...etc.
```

Accordingly, the transition-based declaration of the structures is as follows:

```
trans(Transition,InpConcs,OutComps,InpSigns,OutSigns).
         InpConcs; InpSigns; OutComps; OutSigns = Specified_State*
         Specified_State = n(Coord,State)
```

This solution makes possible the use of the same state names in the various compartments, levels, or scales, because they are identified by the spatial coordinate and the name together.

The declaration of the fictitious example HSC, using the short symbolic names from Table 14.2 is as follows:

```
states([0],[raw_renew,raw_gas,raw_nucl,h2heating,co2,o2]).
states([1],[h2cons,demand,h2carfuel,eleng,hydrogen,h2prod]).
states([2],[h2cons,demand,h2carfuel,raw_land,biomass,biowaste,
eleng,h2contgas,hydrogen,h2prod]).
states([3],[eleng,hydrogen,h2prod,h2cons,demand,h2carfuel]).
states([4],[h2contgas,hydrogen,h2prod,h2cons,demand,h2carfuel,
raw_land,biomass,biowaste,eleng]).

transitions([0],[]).
transitions([1],[t_h2refuel1,t_photovolt1,t_nucl1,t_electrolysis1,
          t_compression1,t_tankertruck11,t_h2heat1]).
```

```
transitions([2],[t_h2refuel2,t_cultivation2,t_photovolt2,t_wind2,
                t_smr2,t_gasify2,t_ferment2,t_electrolysis2,
                t_gassep2,t_compression2,t_tankertruck22,
                t_h2heat2]).
transitions([3],[t_wind3,t_electrolysis3,t_compression3,
                t_h2refuel3,t_tankertruck33,
t_tankertruck31,t_tankertruck32,t_photovolt3,t_hydro3,t_h2heat3]).
transitions([4],[t_smr4,t_gassep4,t_compression4,t_h2refuel4,
                t_tankertruck44,t_tankertruck41,t_tankertruck42,
                t_cultivation4,t_photovolt4,t_hydro4,t_gasify4,
                t_ferment4,t_electrolysis4,t_h2heat4]).

trans(t_h2refuel1,[n([1],h2cons)],[n([1],h2cons),n([1],h2carfuel)],
            [n([1],demand)],[]).
trans(t_h2refuel2,[n([2],h2cons)],[n([2],h2cons),n([2],h2carfuel)],
            [n([2],demand)],[]).
trans(t_h2refuel3,[n([3],h2cons)],[n([3],h2cons),n([3],h2carfuel)],
            [n([3],demand)],[]).
trans(t_h2refuel4,[n([4],h2cons)],[n([4],h2cons),n([4],h2carfuel)],
            [n([4],demand)],[]).
trans(t_tankertruck11,[n([1],h2prod)],[n([1],h2prod),n([1],
            h2cons)],[n([1],h2cons)],[]).
trans(t_tankertruck22,[n([2],h2prod)],[n([2],h2prod),n([2],
            h2cons)],[n([2],h2cons)],[]).
trans(t_tankertruck33,[n([3],h2prod)],[n([3],h2prod),n([3],
            h2cons)],[n([3],h2cons)],[]).
trans(t_tankertruck31,[n([3],h2prod)],[n([3],h2prod),n([1],
            h2cons)],[n([1],h2cons)],[]).
trans(t_tankertruck32,[n([3],h2prod)],[n([3],h2prod),n([2],
            h2cons)],[n([2],h2cons)],[]).
trans(t_tankertruck44,[n([4],h2prod)],[n([4],h2prod),n([4],
            h2cons)],[n([4],h2cons)],[]).
trans(t_tankertruck41,[n([4],h2prod)],[n([4],h2prod),n([1],
            h2cons)],[n([1],h2cons)],[]).
trans(t_tankertruck42,[n([4],h2prod)],[n([4],h2prod),n([2],
            h2cons)],[n([2],h2cons)],[]).
trans(t_compression1,[n([1],hydrogen)],[n([1],hydrogen),
            n([1],h2prod)],[],[]).
trans(t_compression2,[n([2],hydrogen)],[n([2],hydrogen),
            n([2],h2prod)],[],[]).
trans(t_compression3,[n([3],hydrogen)],[n([3],hydrogen),
            n([3],h2prod)],[],[]).
trans(t_compression4,[n([4],hydrogen)],[n([4],hydrogen),
            n([4],h2prod)],[],[]).
trans(t_electrolysis1,[n([1],eleng)],[n([1],eleng),n([1],
              hydrogen)],[n([1],h2prod)],[]).
```

```
trans(t_electrolysis2,[n([2],eleng)],[n([2],eleng),n([2],
              hydrogen)],[n([2],h2prod)],[]).
trans(t_electrolysis3,[n([3],eleng)],[n([3],eleng),n([3],
              hydrogen)],[n([3],h2prod)],[]).
trans(t_electrolysis4,[n([4],eleng)],[n([4],eleng),n([4],
              hydrogen)],[n([4],h2prod)],[]).
trans(t_photovolt1,[],[n([1],eleng)],[],[n([0],raw_renew)]).
trans(t_photovolt2,[],[n([2],eleng)],[],[n([0],raw_renew)]).
trans(t_photovolt3,[],[n([3],eleng)],[],[n([0],raw_renew)]).
trans(t_photovolt4,[],[n([4],eleng)],[],[n([0],raw_renew)]).
trans(t_wind2,[],[n([2],eleng)],[],[n([0],raw_renew)]).
trans(t_wind3,[],[n([3],eleng)],[],[n([0],raw_renew)]).
trans(t_nucl1,[n([0],raw_nucl)],[n([0],raw_nucl),n([1],eleng)],
         [n([1],eleng)],[]).
trans(t_hydro3,[],[n([3],eleng)],[],[n([0],raw_renew)]).
trans(t_hydro4,[],[n([4],eleng)],[],[n([0],raw_renew)]).
trans(t_gassep2,[n([2],h2contgas)],[n([2],h2contgas),n([2],
           hydrogen)],[n([2],h2prod)],[]).
trans(t_gassep4,[n([4],h2contgas)],[n([4],h2contgas),n([4],
           hydrogen)],[n([4],h2prod)],[]).
trans(t_smr2,[n([0],raw_gas)],[n([0],raw_gas),n([2],h2contgas)],
           [n([2],h2contgas)],[]).
trans(t_smr4,[n([0],raw_gas)],[n([0],raw_gas),n([4],h2contgas)],
         [n([4],h2contgas)],[]).
trans(t_ferment2,[n([2],biowaste)],[n([2],biowaste),n([2],
            h2contgas)],[n([2],h2contgas)],[]).
trans(t_ferment4,[n([4],biowaste)],[n([4],biowaste),n([4],
            h2contgas)],[n([4],h2contgas)],[]).
trans(t_gasify2,[n([2],biomass)],[n([2],biomass),n([2],h2contgas)],
          [n([2],h2contgas)],[]).
trans(t_gasify4,[n([4],biomass)],[n([4],biomass),n([4],h2contgas)],
          [n([4],h2contgas)],[]).
trans(t_cultivation2,[n([2],raw_land)],[n([2],raw_land),n([0],co2),
              n([0],o2),n([2],biomass)],[n([2],biomass)],[]).
trans(t_cultivation4,[n([4],raw_land)],[n([4],raw_land),n([0],co2),
              n([0],o2),n([4],biomass)],[n([4],biomass)],[]).
trans(t_h2heat1,[n([1],hydrogen)],[n([1],hydrogen),
          n([0],h2heating)],[],[]).
trans(t_h2heat2,[n([2],hydrogen)],[n([2],hydrogen),
          n([0],h2heating)],[],[]).
trans(t_h2heat3,[n([3],hydrogen)],[n([3],hydrogen),
          n([0],h2heating)],[],[]).
trans(t_h2heat4,[n([4],hydrogen)],[n([4],hydrogen),
          n([0],h2heating)],[],[]).
```

14.4.3 Generation of the Programmable Structure Into a Graphml File

The generation of the editable graphical model utilizes the GraphML description of metaprototypes and the textual definition of the structure discussed above. The generating algorithm first takes the list of state elements and appends the GraphML file with the appropriately modified copies of the state metaprototype. Next, the algorithm takes the list of transition elements and appends the GraphML file with the appropriately modified copies of the transition metaprototype. Finally, the program takes the individual transitions, one after the other, and appends the GraphML file with the description of the edges. The resulting GraphML file contains the two prototypes and the "empty" structure of the whole process model.

The example structure, generated from the two metaprototypes and from the example structure (having already extended by some edited prototypes), can be seen in Fig. 14.4. The GraphML-based graphical model can be edited further by various graph editors (actually yEd Graph Editor).

For better understanding and visualization, the edited process structure is shown in Fig. 14.5.

14.5 PROGRAMMING AND INITIALIZATION OF THE EXAMPLE STRUCTURAL MODEL

The generated GraphML file can be further edited, for example, with the yEd graph editor. The GraphML representation of the process structure supports any graphical modifications and extensions of the model, as well as the addition of editable texts, declaring the local programs, actualizing of initial values and of the various model parameters.

If we start from the automatic generation, then first we have to prepare the editable prototypes from the general metaprototypes. This requires making copies from the metaprototypes, followed by the declaration of the local program of the prototypes. The definition of the state and transition prototype elements needs the creative knowledge of the modelers (advantageously the collaborating field and model experts) about the investigated process. Nevertheless, the implementation is supported by the metaknowledge embedded in the metaprototypes, as well as in the generated process structure.

The variables in prototypes are described by alphanumeric symbols, starting with a capital letter (and written within apostrophes in the graphical interface). The i(), c(), and o() functors are described by a name, by a data type (actually dl), and by a list of the respective variables. Recently, the variables have been described by lists of triplets *d(Identifier, Variable_list, Instructions)*, as was shown in Section 14.4.1.

In the case of the case-specific prototypes, the content of the state and transition elements has to be actualized. This can be done by setting the initial

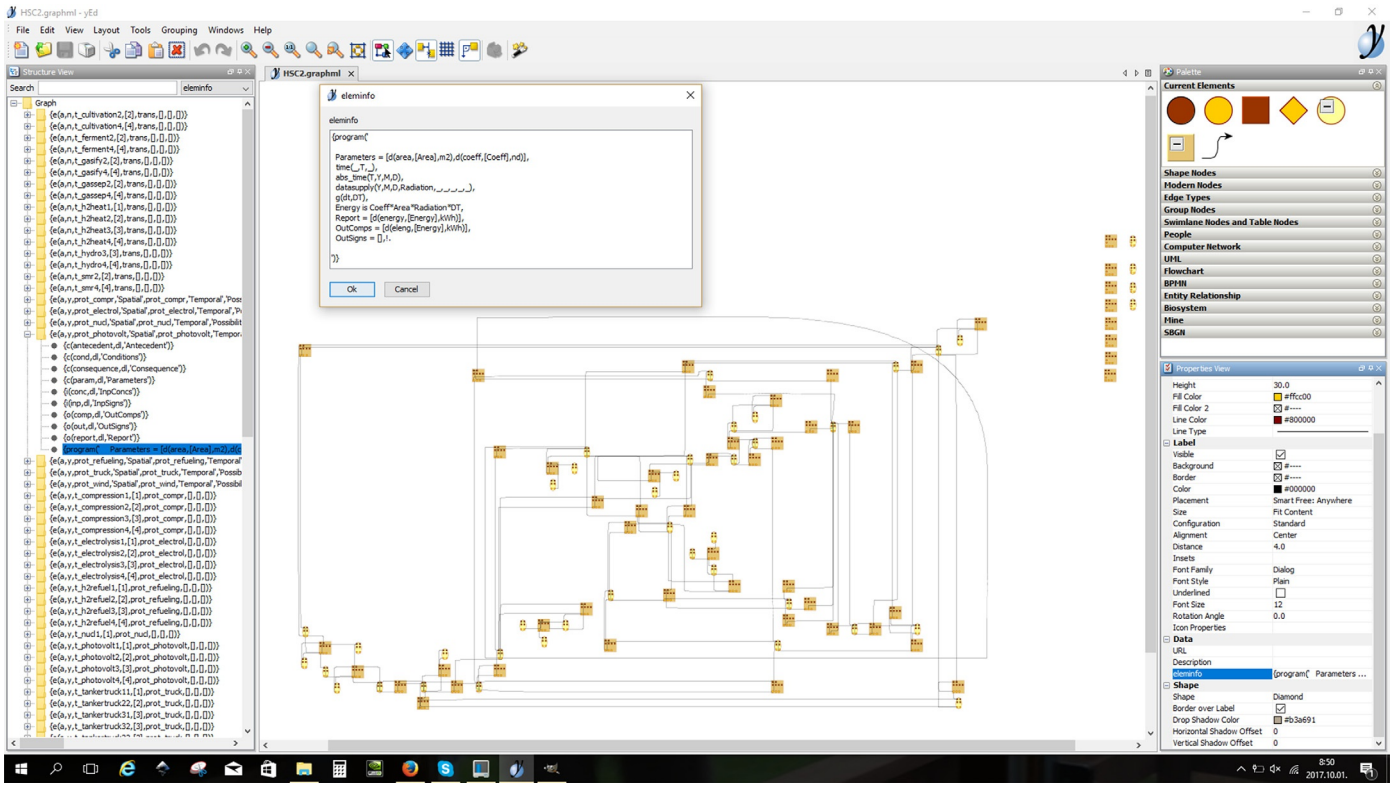


FIG. 14.4 Generated programmable structure of the fictitious example HSC.

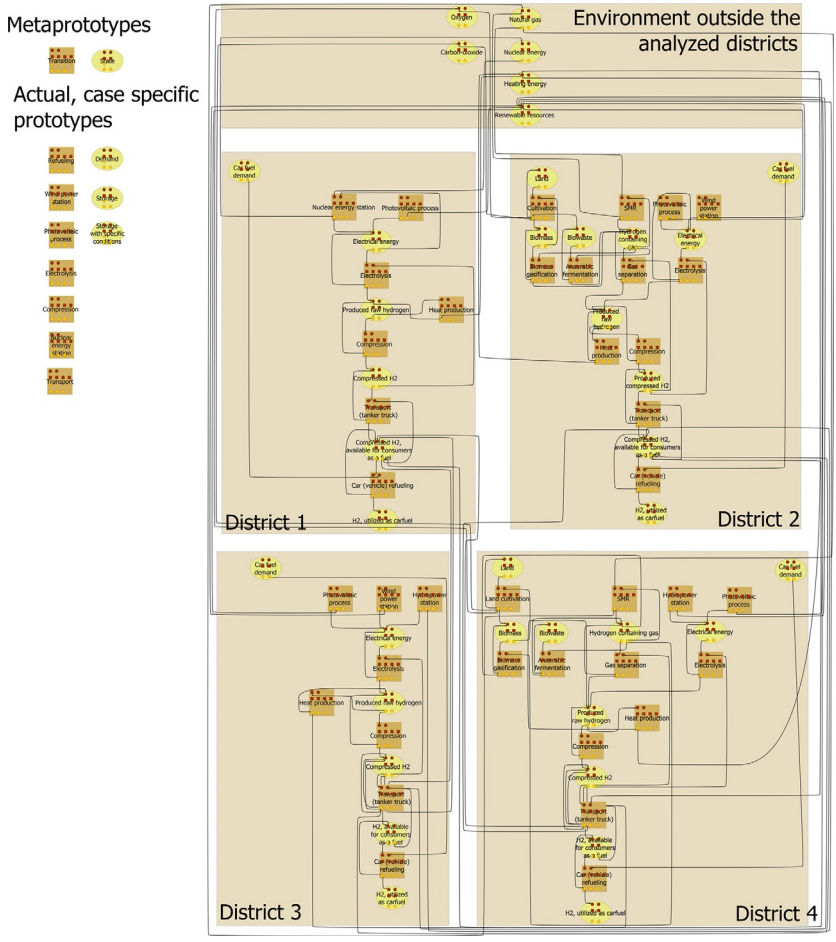


FIG. 14.5 Edited process structure of the fictitious example HSC.

conditions (i.e., the input parameters of the state elements), as well as by defining the parameters (i.e., the actual values of the parameter slots both in state and in transition elements). Also, the name of the prototype that was actually used has to be overwritten in the heading e(.) of each element. This name refers to the local program to be executed for the given element.

The original metaprototypes and the a few actual prototypes of the fictitious example process are also shown in Figs. 14.4 and 14.5. In the illustrative HSC example given, the metaprototypes and prototypes are the following:

- Transition metaprototype: trans;
- Transition prototypes: prot_compr, prot_electrol, prot_nucl, prot_hotovolt, prot_refueling, prot_truck, prot_wind;

- State metaprototype: state;
- State prototypes: prot_demand, prot_storage, prot_condstorage.

In the following, the programming of the prototypes and the initialization of the respective actual elements will be illustrated by simplified examples.

Programming and initialization of transition prototypes and elements.

The transition metaprototype corresponds to the following Prolog clause with an "almost empty" program in the body:

```
v(y,trans,Spatial,trans,Temporal,Possibilities,Evaluations,
[c(antecedent,dl,Antecedent),c(consequence,dl,Consequence),
c(cond,dl,Conditions),c(param,dl,Parameters)],
[i(inp,dl,InpSigns),i(conc,dl,InpConcs)],
[o(comp,dl,OutComps),o(out,dl,OutSigns),o(report,dl,Report)]) :-

 OutComps = [], OutSigns = [], Report = [].
```

When programming of a case-specific transition prototype via the graphical interface, the identifier "trans" has to be replaced by the prototype name (in case of a wind power station "prot_wind"), while the empty program has to be replaced for the respective code, as follows:

```
Parameters = [d(coeff,[Coeff],nd),d(number,[Nu],nd)],
time(_,T,_),
abs_time(T,Y,M,D),
meteo(Y,M,D,_,Windspeed,_,_,_,_),
g(dt,DT),
E is Nu*Coeff*((0.62*Windspeed)**3)*DT,
Report = [d(energy,[E],kWh)],
OutComps = [d(eleng,[E],kWh)],
OutSigns = [].
```

(It is to be noted that in Prolog syntax "is" signifies assignment, "=" refers to unification, and g(dt,DT) calls for the time step.)

Simultaneously in every actual wind power station defining element, the prototype name and the parameters used in the program have to be overwritten (supported by the graph editor, of course), as is signified by bold italics in the following example:

```
a(y,t_wind3,[3],prot_wind,[],[],[],
[c(antecedent,dl,[]),c(consequence,dl,[]),c(cond,dl,[]),
c(param,dl,[d(coeff,[30],nd),d(number,[200],nd)])],
[i(inp,dl,[]),i(conc,dl,[])],
[o(comp,dl,[]),o(out,dl,[]),o(report,dl,[])]).
```

Programming and initialization of state prototypes and elements.

The state metaprototype corresponds to the following Prolog clause with an "almost empty" program in the body:

```
m(y,Spatial,state,Temporal,Possibilities,Evaluations,
[c(param,dl,Parameters),c(cond,dl,Conditions)],
[i(comp,dl,InpComps),i(inp,dl,InpSigns)],
[o(conc,dl,OutConcs),o(out,dl,OutSigns)],) :-
OutConcs = [], OutSigns = [].
```

When programming of a case-specific state prototype via the graphical interface, the identifier "state" has to be replaced by the prototype name (in the case of conditional storage "prot_condstorage"), while the empty program has to be replaced for the respective code, as follows:

```
Conditions = [d(limit,[Limit],Dim),d(destinations,[Place],nd)],
InpComps = [d(Material,[M],Dim)],
needed(Limit,M,Need),
OutConcs = [d(Material,[M],Dim)],
OutSigns = [d(Place,[Need],Dim)].

needed(Limit,M,Need) :-
 Limit > M,
 Need is Limit-M,!.
needed(_,_,0).
```

Simultaneously, in every actual element executed according to this program, the prototype name, the parameters, and the necessary initial values used in the program have to be overwritten, as is signified by bold italics in the following example:

```
p(y,h2prod,[3],prot_condstorage,[],[],[],
[c(param,dl,[]),c(cond,dl,[d(limit,[5000],kg),d(destina-
tions,[electrolysis3],nd)])],
[i(comp,dl,[d(h2prod,[2000],kg)]),i(inp,dl,[])],
[o(conc,dl,[]),o(out,dl,[])]).
```

14.6 INTERPRETATION OF THE EXAMPLE MODEL AND PREPARATION FOR SIMULATION-BASED PROBLEM SOLVING

The programming and initialization of the process structure is followed by its interpretation into the executable dynamic databases of the given model. In this step, the facts and clauses of the executable Prolog model definitions are generated from the edited GraphML file automatically. First, the general interpreter reads the GraphML text, while it generates and saves:

- all of the facts describing the actual state and transition elements in the "user" file,
- all of the facts determining state → transition and transition → state connections in the "user" file, and
- all of the clauses, declaring the program prototypes in the "expert" file.

The resulting user and expert files contain the detailed description of the process model, including the locally executable program prototypes and the actual elements and connections containing the initial data and the case-specific parameters.

Having finished the model generation, the executable code has to be supplemented by the (optionally scale-specific) global data. They can be declared for example by the following global facts:

```
 g(timescale,day).            declares the applied absolute (day, hour,
minute, second) or relative (rel) time (in multiscale application scale-
specific time steps may be applied);
 g(starttime,at(2020,1,1,0,0,0)).  declares  the  absolute  starting
time;
 g(direction,f ).             declares the direction of the simulation
(f = forward, b = backward);
 g(dt,1).                    the initial time step;
 g(dur,3652).                 the duration of the simulation;
 g(ndto,1).                   the time step of the output data recording
(NDTO >= DT);
```

Also, the simulator is prepared for the communication with the necessary optional external databases and applications. For example, in the studied Hydrogen Supply Chain application the model has to communicate:

- with a Geographical Information System, specifying the various districts and locations, as well as assisting with the optionally embedded routing algorithm;
- with a market related database, containing data about the time varying and location specific demand for the hydrogen, and for the optional other products of the model investigated;
- with a meteorological (or climate scenario) database, supplying location and time depending meteorological data for the consideration of the natural resource related processes; and
- with an optional external metaheuristic optimizer (see more detailed in Section 18.8).

An example architecture for combining Programmable Structures with GIS and meteorological databases was published in (Varga et al., 2016b).

14.7 EXECUTION OF THE DYNAMIC SIMULATION OF PROGRAMMABLE STRUCTURE

Dynamic simulation is organized by a general executing kernel, by default comprising the cyclically repeated processing of state elements, state → transition connections, transition elements and transition → state connections.

The simulation starts at the globally determined relative or absolute initial time with the previously initiated data (i.e., input variables of states, parameters of states

and transitions) and with the initial time step. The algorithm checks the Temporal constraints of the given elements and only the allowed procedures are executed. The Temporal constraints may determine prescribed time intervals, discrete points of time, and specific time steps for the given elements, as follows:

```
Temporal = Timing* (may contain any number of timings)
     Timing = t(Start, End, Timelist,TimeStep)
               Start = the prescribed starting time
               End = the prescribed ending time
               Timelist = Time* (may contain any number of discrete
times)
                        Time = prescribed discrete points of time for
execution
               TimeStep = the prescribed time interval of the execution
```

The balance checking of the "model specific conservation law–based measures" helps to recognize the infeasible states (e.g., negative amounts) that may cause serious numerical problems. The sequentially executed state and transition elements, connected by the sequentially executed standalone connections, make it possible to develop robust models, as well as to fine-tune the discretized execution in runtime (e.g., with an automatic time step control, based on feasibility bounds). In some logistic problem solving, the negative values may also be meaningful, and accordingly the time step control is switched off.

During the execution of the local state programs, the state facts are matched with their respective state prototypes. The unification of input and parameter variables is followed by the calculation of the local program, resulting in the values for the output variables.

Analogously, during the execution of the local transition programs, the transition facts are matched with their respective transition prototypes. The unification of input and parameter variables is followed by the calculation of the local program, resulting in the values for the output variables.

The applied architecture makes it possible to save the whole model (together with the actual local and global values) in a file, from which the simulation can be continued later. This feature makes it possible to continue the interrupted simulation with optionally modified parameters to test the effect of various changes. The same option also supports the stepwise communication of the simulation model with external applications (e.g., external control algorithms, additional data sources, etc.).

The simulation models, described by additive measures, are suitable for the causally right reversed (backward) simulation. In this case, the simulation can start from a realistic (or simulated) "future" state with decrementing (backward) time steps, while the increases and extensions of additive measures are replaced with decreases and removals, respectively. Similarly, instead of the decreases and removals of additive measures, increases and extensions are calculated, vice versa. It is to be noted that the functionalities are calculated with the same (i.e., causally right) programs.

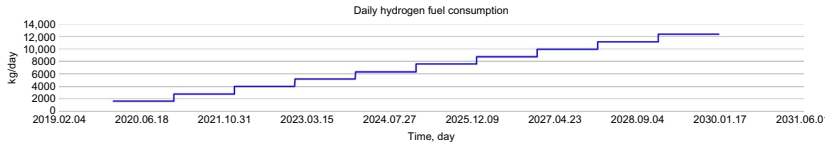


FIG. 14.6 Stepwise increasing demand for hydrogen in the investigated districts.

The detailed output is written in CSV files, to be elaborated further in the case of specifically prepared MS EXCEL workbooks.

In the following we illustrate the results from simulation of a simplified part of the fictitious HSC generated above.

In this case we start from a stepwise, yearly increase of the demand for hydrogen car fueling, as can be seen in Fig. 14.6. Suppose there is an estimated increase in the number of stations and hydrogen-powered cars in the investigated districts in the future time period from 2020 until 2030.

We assume an available capacity of photovoltaic and wind power stations that produce electrical energy, depending on the meteorological situation (solar radiation and wind speed). Unfortunately, the available climate scenarios do not contain data for radiation and wind speed. That is why we used meteorological data from the past 10 years to imitate some realistic fluctuation tendencies. The calculated change of the electrical energy generated in the photovoltaic process is illustrated in Fig. 14.7.

In Fig. 14.8, the contribution of energy generated in the photovoltaic process to the total energy demand of the electrolytic hydrogen production of the necessary car fuel is shown.

Similarly, the calculated change in the electrical energy generated by wind power is illustrated in Fig. 14.9.

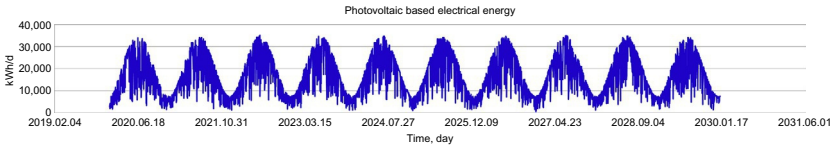


FIG. 14.7 The electrical energy generated in the photovoltaic process in the districts investigated.

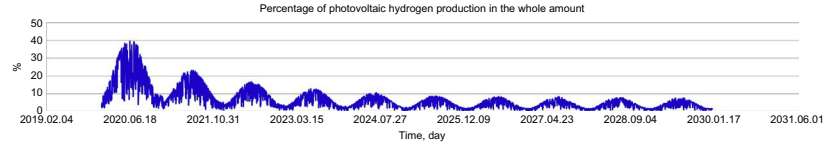


FIG. 14.8 Percentage of photovoltaic-generated energy in the total energy demand of the electrolytic hydrogen production.

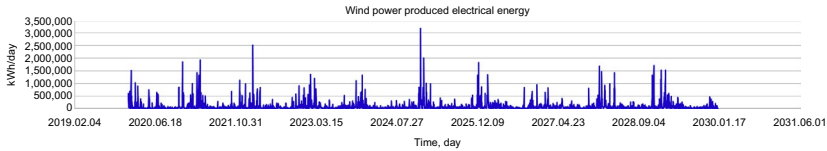


FIG. 14.9 Electrical energy generated by wind power in the investigated districts.

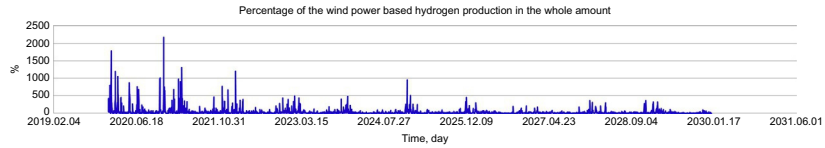


FIG. 14.10 Percentage of electrical energy generated by wind power in the total energy demand of the electrolytic hydrogen production.

In Fig. 14.10 the contribution of energy generated by wind power to the total energy demand of the electrolytic hydrogen production of the necessary car fuel is shown.

It can be seen that in the given districts, wind power can produce more electrical energy than the photovoltaic systems, however, this production fluctuates wildly. In case of lower car fuel demand (in the first years) the solar energy–based renewable resources can produce more energy, and consequently more hydrogen, than the requirement of the car fueling stations. The extraordinary surplus amounts are produced in randomly appearing short periods. The contribution of photovoltaic systems is smoother, with a characteristic seasonal change. Increasing demands for electrolytic hydrogen production requires more renewable energy production or additional electrical energy (e.g., from nuclear power stations, from Steam Methane Reforming, from biomass gasification, etc.)

14.8 REPRESENTATION OF POSSIBILITY (DESIGN) SPACE AND EVALUATIONS IN THE PROGRAMMABLE STRUCTURE

The robust simulation framework makes possible *dynamic model-based (sub)-optimization*, for example, with various externally implemented metaheuristic methods (e.g., Genetic Algorithm). The basic principle, in accordance with the engineering way of thinking, is that instead of searching for an exact global optimum with a simplified model, it might be better to generate suboptimal solutions with a more detailed model. This principle was successfully applied for the solution of various design problems by the previous version of the Direct Computer Mapping–based simulator (e.g., Csukas et al., 2013).

Using Programmable Structures, the possibility (design) space and the evaluations can be adequately defined within the model elements. This extended

simulation model can communicate with various external optimizing tools by means of special evaluating and configuring connections. The evaluations (i.e., calculated objective functions for the optimizing program) come from the output variables of the evaluating elements, while the new configurations (i.e., the formal suggestions of the optimizer) are processed by the configuring elements of the simulation model. The advantage of this solution is that interfacing between the detailed model and the external optimizer can be organized within the model, with the knowledge of the model-specific details. At the same time, the optimizer may work with a number of evaluating objectives and with an abstract possibility (design) space, formally. The collaboration between the simulator and the optimizer is determined by the respective evaluation feedback.

In *combining Programmable Structure of Hydrogen Supply Chains with metaheuristic optimizers*:

- the possibility (design) space and the components of evaluations are defined within the state and transition elements, while the model is extended with special configuring and evaluating elements;
- the simulator initializes the formal connection with optional optimizers at the beginning of the run;
- the (optionally in-parallel running) model(s) organize the repeated configuration and evaluation of the communication with the metaheuristic optimizer.

The *uncertainty* of model parameters and/or evaluations (e.g., cost coefficients) can be taken into consideration with an additional objective for minimizing the standard deviation of the (e.g., economic) objective, using a set of range-based, randomly generated parameters for each scenario. This method was applied for the optimization of a methane producing anaerobic fermentor of sugar beet slices (Varga, 2009).

14.8.1 Embedding Possibilities in the State and Transition Elements

The *Possibilities*, belonging to a given element, can be declared by the following data set:

```
Possibilities = Possibility*
Possibility = pos(PosID,VariableIdentifier,Type,Mode,PossibleValues,
              SuggestedValue)
```

where:

PosID = identifies the given modification in the possibility (design) space,
VariableIdentifier = identifies the variable to be modified in the given model element,

Type = determines the type of the variable (symbol, integer, real or timing).
Mode = defines the method of modification (select or range),
PossibleValues = defines the possibilities (alternatives) for the given variable,
SuggestedValue = the actually proposed upgraded value (initialized by a default value).

The most important configurable characteristics of model elements are the followings:

- existence of state and transition elements (Type = symbol);
- initial input measures of the state elements (Type = real);
- initial input signs of the state elements (Type = symbol);
- state parameters (Type = symbol, integer or real);
- transition parameters (Type = symbol, integer or real);
- timing of transitions: (Type = timing); etc.

In accordance with the above model properties, the possible alternatives can be defined, as follows:

- by a set of symbols to select an actual one (Mode = select);
- by a set of integers to select an actual one (Mode = select);
- by a range of integers, and its initial resolution to determine the prescribed granularity that can be refined stepwise (Mode = range);
- by a range of real numbers, and its initial resolution to determine the prescribed granularity that can be refined stepwise (Mode = range);
- by a set of possible timings in sense of Eq. 6 to select an actual one (Mode = select or range); etc.

In case of (Mode = select) PossibleValues are defined by the set of possible values, while in case of (Mode = range).

```
PossibleValues = [Min,Max,Resolution]
```

where Min = the minimal value, Max = the maximal value and Resolution = the suggested initial resolution of the range (that can be modified by the external optimizer).

At the beginning of the optimizing cycle, the domain of the possibility (design) space is translated for the external optimizer. It should be emphasized that the optimizer does not know about the meaning of the data to be modified and considers only their formal representation. Accordingly, an actual Possibility is transformed into the tuple.

```
(PosID,Number,Min,Max,Resolution)
```

where PosID = the identifier of the change, Number = the number of possible values or 0, Min = the minimal value or 0, Max = the maximal value or 0 and Resolution = the suggested initial resolution within the range or 0.

The decisions from the optimizer come in the form of:

```
(PosID,Ordinal_number,SuggestedValue)
```

where Ordinal_number = the respective element of the set or 0, Suggested Value = the suggested value from the range or 0, while these formal messages are interpreted according to PosID.

14.8.2 Embedding Elementary Evaluations in the State and Transition Elements

In addition to the very uncertain economic objectives, the optimal design and operation of an HSC requires multiple natural objectives (utilized solar energy, carbon-dioxide balance, hydrogen production of biological wastes, the food-related part of land use, etc.). Accordingly, evaluations have to be embedded into the elements of the more detailed dynamic models.

The *Evaluations*, belonging to a given element, can be declared by the following data set:

```
Evaluations=Evaluation*.
Evaluation=ev(EvalID,EvaluationName,VariableIdentifier,Mode,
Measurements).
```

where:

EvalID = an integer identifier of the corresponding objective (used by all of the respective measurements);
EvaluationName = a symbol, describing the objective function;
VariableIdentifier = a simulated variable corresponding to the measured data or nil;
Mode = mode of the use (e.g., ident, max, min, etc.); and.
Measurements = set of ($Time_i$,$Value_i$) pairs for the respective variable, while Value means the measured data at Time.

The individual evaluations are executed by the objective defining transitions that calculate the objective functions individually. Considering the above kinds of objectives, there are three modes of evaluations, as follows:

- for identification according to the supplemented measurements (Mode = ident);
- for maximization of an objective function (Mode = max);
- for minimization of an objective function (Mode = min).

At the beginning of the optimizing cycle, the identifiers of objectives are transformed for the external optimizer. It should be emphasized that the optimizer does not know about the meaning of the objective functions and considers only their formal representation. Accordingly, an Evaluation for the optimizer is described by the pair of.

```
(EvalID,EvaluationName).
```

The most important data of the model elements, used for objective evaluations are the following:

- intensive properties at the output of the state elements;
- special output data of the transition elements: for example, the reported rate of a given transformation or transportation; and
- state and transition parameters for the calculation of objective functions.

The individual evaluating components, identified by an EvalID, are collected during and/or at the end of simulation by the respective evaluating objectives. At the end of the individual simulations these special evaluating elements calculate the values of the objective functions and forward them to the optimizer in the form of:

```
(EvalID,Calculated_Objective)
```

where EvalID = the identifier of the objective and Calculated_evaluation = the value of this objective function after the simulation.

14.9 CONCLUSIONS AND FURTHER WORK

Besides the well-developed mathematical programming–based optimization methods, the design and operation of the large-scale, long-term processes of Hydrogen Supply Chains might also require easily extensible, generic dynamic simulation. We have introduced a nonconventional modeling and simulation methodology, developed in other multidisciplinary fields.

Keeping in mind the challenges of Hydrogen Supply Chain design and operation, we explained Programmable Structure for experts of the given field to promote possible future collaboration. Programmable Structure of process models can be generated from the description of a process network (optionally geographically determined and multiscale) and from two general functional metaprototypes. The case-specific, actual functional prototypes are copied from the metaprototypes, and they are associated with locally executable declarative programs in a graphical editor. Simultaneously, the actual state and transition elements are parameterized and initialized concerning their case-specific prototypes. The execution of the programmed structures is solved by a general-purpose kernel program, while the actual model elements are executed according to their prototypes. In the proposed model representation, the detailed description of possibility (design) space, as well as the components of the evaluating functions may be embedded in the state and transition elements.

The methodology has been illustrated with a fictitious and simple example in the context of Hydrogen Supply chains.

The major points of the work planned for the future, in collaboration with HSC experts, are the following:

- to test the framework for a realistic case study;

- to investigate the application of repeated forward/backward simulation based balancing algorithms;
- to combine Programmable Structures with metaheuristic optimizers for the multiobjective optimization of HSC design and operation.

ACKNOWLEDGMENT

Work was supported by the TÉT_16-1-2016-0116 French-Hungarian bilateral program.

REFERENCES

Csukas, B., 1998. In: Simulation by direct mapping of the structural models onto executable programs. AIChE Annual Meeting 1998, Miami, Paper 239/9.

Csukas, B., Balogh, S., Kovats, S., Aranyi, A., Kocsis, Z., Bartha, L., 1999. Process design by controlled simulation of the executable structural models. In: Comput. Chem. Eng. 23 (Suppl), 569–572.

Csukas, B., Varga, M., Balogh, S., 2011. Direct computer mapping of executable multiscale hybrid process architectures.Proceedings of Summer Simulation Multiconference'2011, Den Haag, The Netherlands, 2011.06.26–2011.06.29, pp. 87–95. ISBN: 1-56555-345-4.

Csukas, B., Varga, M., Miskolczi, N., Balogh, S., Angyal, A., Bartha, L., 2013. Simplified dynamic simulation model of plastic waste pyrolysis in laboratory and pilot scale tubular reactor. Fuel Process. Technol. 106, 186–200. https://doi.org/10.1016/j.fuproc. 2012.07.024.

Dagdougui, H., 2012. Models, methods and approaches for the planning and design of the future hydrogen supply chain. Int. J. Hydrog. Energy 37, 5318–5327. https://doi.org/10.1016/j.ijhydene.2011.08.041.

De-Leon Almaraz, S., 2014. Multi-objective optimisation of a hydrogen supply chain. Ph.D. Thesis, Université de Toulouse.

De-Leon Almaraz, S., Azzaro-Pantel, C., 2017. Chapter 4—Design and optimization of hydrogen supply chains for sustainable future. In: Scipioni, A., Manzardo, A., Ren, J. (Eds.), Hydrogen Economy. Academic Press, Cambridge, MA, pp. 85–120.

Govindan, K., Fattahi, M., Keyvanshokooh, E., 2017. Supply chain network design under uncertainty: a comprehensive review and future research directions. Eur. J. Oper. Res. 263, 108–141. https://doi.org/10.1016/j.ejor.2017.04.009.

Hydrogen Council, 2017. How hydrogen empowers the energy transition. Available from: http://hydrogeneurope.eu/wp-content/uploads/2017/01/20170109-HYDROGEN-COUNCIL-Vision-document-FINAL-HR.pdf. Accessed 21 September 2017.

Maryam, S., 2017. Review of modelling approaches used in the HSC context for the UK. Int. J. Hydrog. Energy 42 (39), 24927–24938. https://doi.org/10.1016/j.ijhydene.2017.04.303.

Nurjanni, K.P., Carvalho, M.S., Costa, L., 2017. Green supply chain design: a mathematical modeling approach based on a multi-objective optimization model. Int. J. Prod. Econ. 183, 421–432. https://doi.org/10.1016/j.ijpe.2016.08.028.

Petri, C.A., 1962. Kommunikation Mit Automaten (Communication with Automata). Ph.D. Thesis, University of Bonn.

Petri, C.A., 1977. General Net Theory.Proc. of 1976 Joint IBM/Univ. of Newcastle upon Tyne Seminar on Computing System Design, Newcastle upon Tyne, Computing Laboratory, 1977. pp. 131–169. http://homepages.cs.ncl.ac.uk/brian.randell/GeneralNetTheory.pdf.

Samsatli, S., Staffell, I., Samsatli, N.J., 2016. Optimal design and operation of integrated wind-hydrogen-electricity networks for decarbonising the domestic transport sector in great Britain. Int. J. Hydrog. Energy 41, 447–475. https://doi.org/10.1016/j.ijhydene.2015.10.032.

Temesvári, K., Aranyi, A., Csukas, B., Balogh, S., 2004. Simulated moving bed separation of a two components steroid mixture. Chromatographia 60, 189–199.

Varga, M., 2009. Economic optimization of sustainable complex processes (model based optimization under uncertain cost parameters for industrial scale anaerobic fermentation of sugar beet slice). PhD Thesis, Kaposvar University, Kaposvar (in Hungarian).

Varga, M., Csukas, B., 2017. Generation of extensible ecosystem models from a network structure and from locally executable programs. Ecol. Model. 364C, 25–41.

Varga, M., Csukás, B., Balogh, S., 2012. Transparent Agrifood Interoperability, Based on a Simplified Dynamic Simulation Model. In: Mildorf, T., Charvat, K. (Eds.), ICT for Agriculture, Rural Development and Environment: Where We Are? Where We will Go?. Czech Centre for Science and Society, Prague. ISBN: 978-80-905151-0-9, pp. 155–174.

Varga, M., Balogh, S., Wei, Y., Li, D., Csukas, B., 2016a. Dynamic simulation based method for the reduction of complexity in design and control of recirculating aquaculture systems. Inf. Process. Agric. 3, 146–156. https://doi.org/10.1016/j.inpa.2016.06.001.

Varga, M., Balogh, S., Csukas, B., 2016b. An extensible, generic environmental process modelling framework with an example for a watershed of a shallow lake. Environ. Model. Softw. 75, 243–262. https://doi.org/10.1016/j.envsoft.2015.10.022.

Varga, M., Prokop, A., Csukas, B., 2017. Biosystem models, generated from a complex rule/reaction/influence network and from two functionality prototypes. Biosystems 152, 24–43. https://doi.org/10.1016/j.biosystems.2016.12.005.

Wang, J., Yin, Y., 2017. Principle and application of different pretreatment methods for enriching hydrogen-producing bacteria from mixed cultures. Int. J. Hydrog. Energy 42 (8), 4804–4823. https://doi.org/10.1016/j.ijhydene.2017.01.135.