AN INTRODUCTION TO MULTI-PARADIGM MODELLING AND SIMULATION

Hans Vangheluwe
School of Computer Science
McGill University, Montréal
Québec, Canada
hv@cs.mcgill.ca

Juan de Lara
E.T.S. de Informática
Universidad Autonóma de Madrid
Madrid, Spain
Juan.Lara@ii.uam.es

Pieter J. Mosterman
Simulation and Real-Time Technologies
The MathWorks, Inc.
Natick, MA, USA
pieter_j_mosterman@mathworks.com

ABSTRACT

Modelling and simulation are becoming increasingly important enablers in the analysis and design of complex systems. To tackle problems of ever increasing complexity, modelling and simulation research is shifting from simulation techniques to modelling methodology and technology. In this article, the emerging field of Computer Automated Multi-Paradigm Modelling is presented. Multi-paradigm modelling adresses and integrates three orthogonal directions of research:

1. multi-formalism modelling, concerned with the coupling of and transformation between models described in different formalisms,
2. model abstraction, concerned with the relationship between models at different levels of abstraction, and
3. meta-modelling, concerned with the description (models of models) of classes of models, which allows formalism specification.

The article first introduces the general concepts of Modelling and Simulation theory, and explains how rigourous application thereof provides a sound basis for the meaningful exchange and re-use of knowledge about the behaviour of complex systems. The representation of models in diverse formalisms, at different levels of abstraction, and the (behaviour-conserving) transformation between the formalisms is demonstrated.

1 MODELLING AND SIMULATION

At a first glance, it is not easy to characterize modelling and simulation. Certainly, a variety of application domains such as fluid dynamics, energy systems, and logistics management make use of it in one form or another. Depending on the context, modelling and simulation is often seen as a sub-set of Systems Theory, Control Theory, Numerical Analysis, Computer Science, Artificial Intelligence, or Operations Research. Increasingly, modelling and simulation *integrates* all of the above disciplines. As a paradigm, it is a way of representing problems and thinking about them as much as a solution method. The problems cover the analysis and design of complex dynamical systems. In analysis, abstract models are built inductively from observations of a real system. In design, models deductively derived from a priori knowledge are used to build a system, satisfying certain design goals. Often, an iterative combination of analysis and design is needed to solve real problems. Though the focus of modelling and simulation is on the time-varying behaviour of dynamical systems, static systems (such as Entity-Relationship (ER) models, described in the Unified Modeling Language (UML) (Rumbaugh et al., 1999)) are a limit-case. Both physical (obeying conservation and constraint laws) and non-physical (informational, such as software) systems and their interactions are studied.

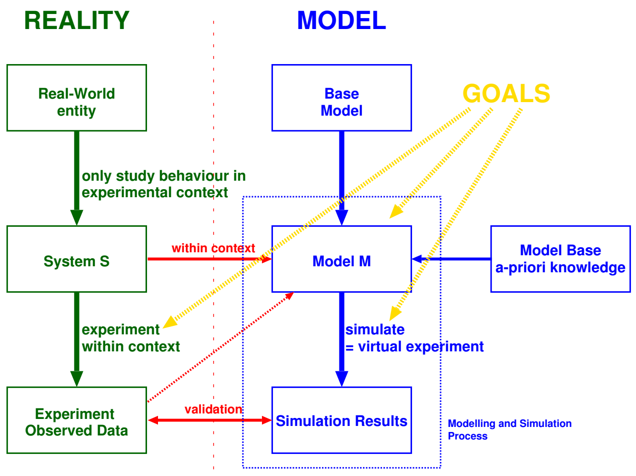
REALITY
MODEL
GOALS
Real-World entity
Base Model
only study behaviour in experimental context
System S
within context
Model M
Model Base a-priori knowledge
experiment within context
simulate = virtual experiment
Experiment Observed Data
validation
Simulation Results
Modelling and Simulation Process

Figure 1: Modelling and Simulation concepts.

1.1 Concepts

Figure 1 presents modelling and simulation concepts as introduced by Zeigler (Zeigler, 1984b; Zeigler et al., 2000).

Real World Entity or object can exhibit widely varying behaviour depending on the context in which it is studied as well as the aspects of its behaviour which are under study.

Base Model is a hypothetical, abstract representation of the object's properties, in particular, its behaviour, which is valid in all possible contexts, and describes all the object's facets.

System is a well defined object in the Real World under specific conditions, only considering specific aspects of its behaviour.

Experimental Frame (EF) describes a limited set of circumstances under which a system (real or model) is to be observed or subjected to experimentation. As such, the Experimental Frame reflects the *objectives* of the experimenter who performs experiments on a real system or, through simulation, on a model.

(Lumped) Model (not to be confused with a lumped parameter model (Cellier, 1991)) is an abstract representation of a system within the context of a given Experimental Frame. Usually, certain properties of the system's structure and/or behaviour are reflected by the model (within a certain range of accuracy).

Experimentation is the act of carrying out an experiment. An experiment may interfere with system operation (influence its input and parameters) or it may not. As such, the experimentation environment may be seen as a system in its own right (which may itself be modelled in a lumped model). Also, experimentation involves observation. Observation yields *measurements*.

Simulation of a lumped model in a certain formalism (such as Petri Net, Differential Algebraic Equations (DAE) or Bond Graph) computes the dynamic input/output behaviour. Simulation may use symbolic as well as numerical techniques. Simulation, which mimics the real-world experiment, can be seen as *virtual experimentation*. Whereas the goal of modelling is to *meaningfully* describe a system, presenting information in an understandable, re-usable way, the aim of simulation is to be *fast and accurate*. Symbolic techniques are often favoured over numerical ones as they allow the generation of classes of solutions rather than just a single one (*e.g.,* $sin(x)$ as a solution to the harmonic equation as opposed to one single approximate trajectory solution). Furthermore, symbolic optimizations have a much larger impact than numerical ones thanks to their global nature. Crucial to the System–Experiment/Model–Virtual Experiment scheme is that there is a *homomorphic* relation between model and system: building a model of a real system and subsequently simulating its behaviour should yield the same results as performing a real experiment followed by observation and codifying the experimental results.

Verification is the process of checking the consistency of a simulation program with respect to the lumped model it is derived from.

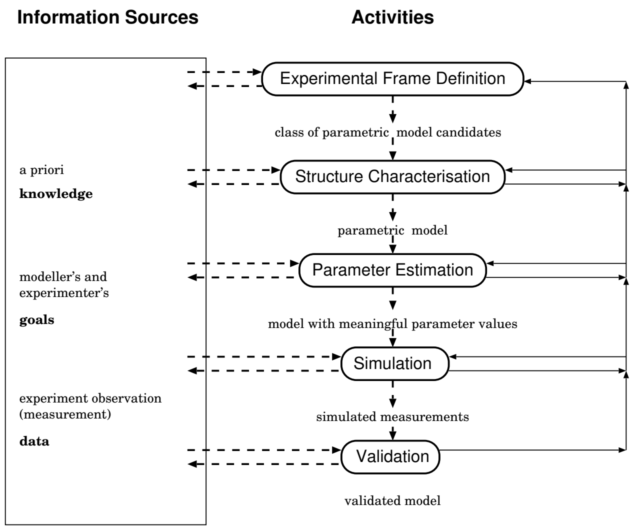
Information Sources
Activities
Experimental Frame Definition
class of parametric model candidates
a priori
knowledge
Structure Characterisation
parametric model
modeller's and experimenter's
goals
Parameter Estimation
model with meaningful parameter values
Simulation
experiment observation (measurement)
data
simulated measurements
Validation
validated model

Figure 2: Model-based systems analysis.

Validation is the process of comparing experiment *measurements* with *simulation results* within the context of a certain Experimental Frame (Balci, 1997). When comparison shows differences, the formal model built may not correspond to the real system. A large number of matching *measurements* and *simulation results*, though increasing confidence, does *not prove* validity of the model however. For this reason, Popper has introduced the concept of *falsification* (Magee, 1985), the enterprise of trying to falsify or disprove a model. It is to be noted that the correspondence in generated behaviour between a system and a model will only hold within the limited *context* of the Experimental Frame. Consequently, when using models to exchange information between human or computer agents, a model must always be matched with an Experimental Frame before use. Conversely, a model should never be developed without simultaneously developing its Experimental Frame. This requirement has its repercussions on the design of model representation languages.

1.2 The Modelling and Simulation Process

The use of simulation has proven to be invaluable in the study of complex systems. The simulation activity is part of the larger model-based systems analysis enterprise. A framework for these activities is depicted in Figure 2. The Framework starts by identifying an Experimental Frame. As mentioned above, the frame represents the experimental conditions under which the modeller wants to investigate the system. As such, it reflects the modeller's goals and questions. Based on a frame, a class of matching models can be identified. Through structure characterization, the appropriate model structure is selected based on a priori knowledge and measurement data. Subsequently, during model calibration, parameter identification (or estimation)

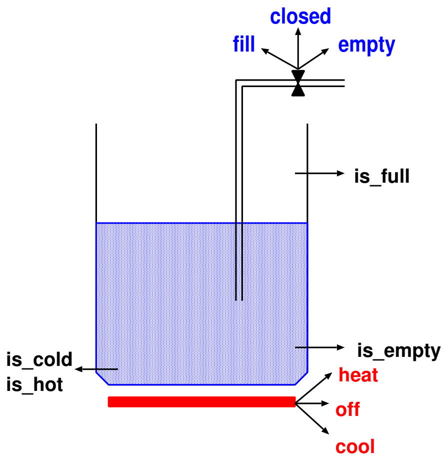

Figure 3: T,l controlled liquid in a vessel.

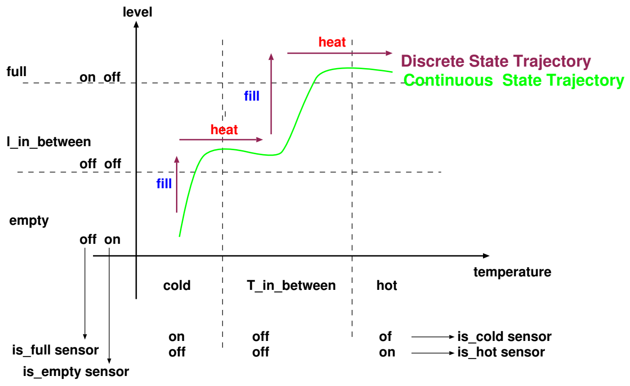

Figure 4: Trajectories.

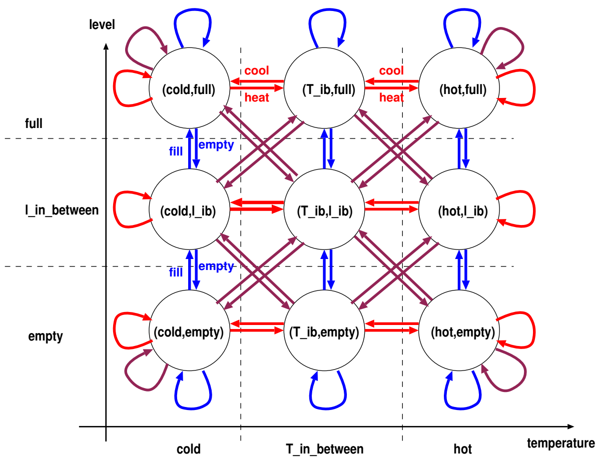

Figure 5: FSA formalism.

yields optimal parameter values for reproducing a set of measurement data. Using the identified model and parameters, simulation allows one to mimic the system behavior (virtual experimentation). The question remains however whether the model has predictive validity: is it capable not only of reproducing data which was used to choose the model and to identify parameters, but also of predicting new behavior ? In Figure 2, one notices how each step in the modelling process may introduce errors. As indicated by the feedback arrows, a model has to be corrected once falsified.

2 ABSTRACTION AND FORMALISM

Abstract models of system behaviour can be described at different levels of abstraction or detail as well as by means of different formalisms. The particular formalism and level of abstraction depends on the background and goals of the modeller as much as on the system modelled. As an example, a temperature and level controlled liquid in a vessel is considered. This is a simplified version of the system described in (Barros et al., 1998), where structural change is the main issue. On the one hand, the liquid can be heated or cooled. On the other hand, liquid can be added or removed. In this simple example phase changes are not considered. The system behaviour is completely described by the following Ordinary Differential Equation (ODE) model:

$$\left\{\begin{array}{rcl} \frac{dT}{dt} & = & \frac{1}{l}[\frac{W}{c\rho A} - \phi T] \\ \frac{dl}{dt} & = & \phi \text{ if } 0 < l < H \text{ else } 0 \\ is\_low & = & (l < l_{low}) \\ is\_high & = & (l > l_{high}) \\ is\_cold & = & (T < T_{cold}) \\ is\_hot & = & (T > T_{hot}) \end{array}\right.$$

The inputs are the filling (or emptying if negative) flow rate ϕ, and the rate W at which heat is added (or removed if negative). This system is parametrized by A, the cross-section surface of the vessel, H, its height, c, the specific heat of the liquid c, and ρ, the density of the liquid. The state of the system is characterized by variables T, the temperature and l, the level of the liquid. The system is observed through threshold output sensors $is\_low, is\_high, is\_cold, is\_hot$. Given input signals, parameters, and a physically meaningful initial condition (T_0, l_0), simulation of the behaviour yields a continuous state trajectory as depicted in Figure 4. By means of the binary (on/off) level and temperature sensors introduced in the differential equation model, the state-space may be discretized. The inputs can be abstracted to heater heat/cool/off and pump fill/empty/closed. At this level of abstraction, a Finite State Automaton representation (with 9 possible states) of the dynamics of the system as depicted in Figure 5 is most appropriate. Though at a much higher level of abstraction, this model is still able to capture the essence of the system's behaviour. In particular, there is a *behaviour morphism* between both models: model discretization (from ODE to FSA) followed by simulation yields the same result as simulation of the ODE followed by discretization.

2.1 Systems Theory

In general systems theory (Wymore, 1967), a causal (output is the consequence of given inputs), deterministic (a known input will lead to a unique output) system model SYS is defined. It is a template for a plethora of different formalisms such as Ordinary Differential Equations, Finite State Automata, Difference Equations, Petri Nets, *etc.* Its general form is

$$SYS \equiv \langle \mathcal{T}, X, \Omega, Q, \delta, Y, \lambda \rangle$$

$\mathcal{T}$	time base
X	input set
$\omega : \mathcal{T} \to X$	input segment
Q	state set
$\delta : \Omega \times Q \to Q$	transition function
Y	output set
$\lambda : Q \to Y$	output function

$$\forall t_x \in [t_i, t_f]$$

$$\delta(\omega_{[t_i,t_f]}, q_i) = \delta(\omega_{[t_x,t_f]}, \delta(\omega_{[t_i,t_x]}, q_i))$$

The time base $\mathcal{T}$ is the formalisation of the independent variable time, also known as the indexing variable (Nance, 1981). The input set describes all possible allowed inputs (possibly a product set). An input represents input during a time-interval. The history of system behaviour is condensed into a *state*. The dynamics is described in a transition function which takes a current state, and applies an input segment $\omega \in \Omega$ to it to obtain a new state. The system may generate output. The output is obtained as a function λ of the state. State and transition function must obey the composition or transitivity property. This property, whereby a transition over a time interval $[t_i, t_f]$ can always be split into a composition of transitions over arbitrary sub-intervals, is the basis of all model simulators. As the output function is described separately, efficient simulators will only invoke it when the user desires to observe output. As SYS is a template for different formalisms, it is possible to describe both models of the vessel example. In the ODE case, the time base is continuous ($\mathbb{R}$). The transition function is written in integral form. Different numerical approximations of the integral can be used in the implementation of an abstract simulator.

$$SYS^{ODE}_{VESSEL} = \langle \mathcal{T}, X, \Omega, Q, \delta, Y, \lambda \rangle$$

$\mathcal{T} = \mathbb{R}$
$X = \mathbb{R} \times \mathbb{R} = \{(W, \phi)\}$
$\omega : \mathcal{T} \to X$
$Q = \mathbb{R}^+ \times \mathbb{R}^+ = \{(T, l)\}$
$\delta : \Omega \times Q \to Q$
$\delta(\omega_{[t_i,t_f]}, (T(t_i), l(t_i))) =$

$$(T(t_i) + \int_{t_i}^{t_f} \frac{1}{l(\alpha)} [\frac{W(\alpha)}{c\rho A} - \phi(\alpha) T(\alpha)] d\alpha,\ l(t_i) + \int_{t_i}^{t_f} \phi(\alpha) d\alpha)$$

$Y = \mathbb{B} \times \mathbb{B} \times \mathbb{B} \times \mathbb{B} = \{(is\_low, is\_high, is\_cold, is\_hot)\}$
$\lambda : Q \to Y$

Q	$\mathcal{T}$: **Continuous**	$\mathcal{T}$: **Discrete**	$\mathcal{T}$: $\{NOW\}$
Continuous	DAE	Difference Eqns.	Algebraic Eqns.
Discrete	Discrete-event Naive Physics	FSA Petri Nets	Integer Eqns.

Table 1: Formalism classification.

$\lambda(T, l) = ((l < l_{low}), (l > l_{high}), (T < T_{cold}), (T > T_{hot}))$

At a higher level of abstraction, we have represented time as a discrete integer index. The transition function lists all possible state transformations.

$$SYS^{FSA}_{VESSEL} = \langle \mathcal{T}, X, \Omega, Q, \delta, Y, \lambda \rangle$$

$\mathcal{T} = \mathbb{N}$
$X = \{heat, cool, off\} \times \{fill, empty, closed\}$
$\omega : \mathcal{T} \to X$
$Q = \{cold, T_{between}, hot\} \times \{empty, l_{between}, full\}$
$\delta : \Omega \times Q \to Q$
$\delta((off, fill), (cold, empty)) = (cold, l_{between})$
$\delta((off, fill), (cold, l_{between})) = (cold, full))$
$\delta((off, fill), (cold, full))) = (cold, full))$

$$\vdots$$

$\delta((heat, fill), (hot, full))) = (hot, full))$
$Y = \mathbb{B} \times \mathbb{B} \times \mathbb{B} \times \mathbb{B}$
$\lambda : Q \to Y$
$\lambda(T, l) = ((l = low), (l = high), (T = cold), (T = hot))$

Apart from the two causal, deterministic formalisms presented above, a host of other formalisms are in use. In particular, through time-scale or parameter abstraction, reality is often represented by means of *discrete-event models*. In these models, time evolves continuously ($\mathcal{T} = \mathbb{R}$), but the state of the system only changes at a finite number of points in time in a bounded time-interval. These times are called event-times. A number of discrete-event formalisms, called *world views* were constructed. World views range from Event Scheduling which expresses events and their consequences explicitly, focussing on the simulation efficiency, over Activity Scanning and the Three Phase Approach which emphasize changing conditions in the system, to Process Interaction which represents the system in terms of interacting processes (Balci, 1988). Zeigler (Zeigler, 1984a) developed the DEVS discrete-event system specification. It allows other discrete-event formalisms to be expressed in terms of DEVS.

2.2 Formalism Classification

Formalism classification based on the general system model structure in Table 1 shows how different instantiations of that structure lead to different formalisms. For continuous models, classification according to physical domains such

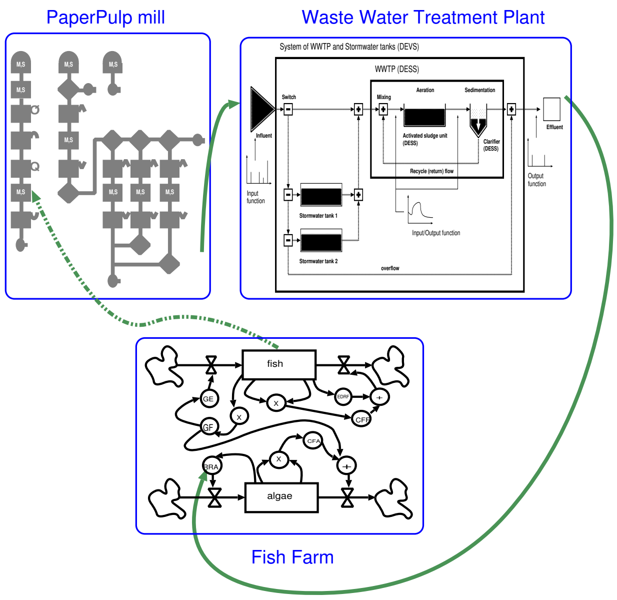

Figure 6: Complex system example.

as mechanical, electrical, and hydraulical, is meaningful. The variety of classifications leads to the insight that ultimately, one should picture a vast formalism-space and classify that space according to various criteria. Different criteria will lead to different equivalence classes. The *SYS* time base and state set allow for a high-level classification of formalisms. Whithin the discrete-event realm, the world views provide a further means of classification. Note how many languages and tools correspond to each single formalism.

Though quite generic, the formalisms presented do not describe *non-causal* models. When describing physical systems, non-causal models describe conservation laws and constraints without imposing a computational causality. Non-causal modelling formalisms and how they enable meaningful model re-use are described in (Cellier, 1991).

3 MULTI-FORMALISM MODELLING

Complex systems are characterized, not only by a large *number* of components, but also by the *diversity* of the components. One of the observations of the European Commission's ESPRIT Basic Research Working Group 8467 (Vangheluwe and Vansteenkiste, 1996) "Simulation for the Future: New Concepts, Tools and Applications" was that for the analysis and design of such complex systems, it is no longer sufficient to study the diverse components separately, using the specific formalisms these components were modelled in. Rather, it is necessary to answer questions about properties –most notably behaviour– of the *whole* system. Also, to communicate knowledge about these systems, one must deal with the diversity of the information.

3.1 A Complex System Example

To focus the attention, Figure 6 presents a complex system. The complexity lies in the diversity of the different components, both in abstraction level and in formalism used:

- A paper and pulp mill produces paper from trees with polluted water as a side-effect. This system is modelled as a process interaction discrete-event scheduling system (in particular, in GPSS (Gordon, 1996)).
- A Waste Water Treatment Plant (WWTP) purifies the polluted effluent from the mill. Some solid waste is taken to a landfill. The partially purified water flows into a lake. This system is modelled using Differential Algebraic Equations (DAEs) describing the biochemical reactions in the WWTP.
- A Fish Farm grows fish in the lake. The fish feed on algae which are highly sensitive to polluted water. The water is also used for a tree plantation which supplies the paper mill. This eco-system is modelled using the Forrester System Dynamics formalism. The dotted feedback arrow from the fish farm to the paper mill indicates the possible disastrous impact of poisoned fish on the productivity of workers in the mill.

It is obvious that decision-making for this system will require understanding of the behaviour of the overall system. Studying the individual components will not suffice. The complexity of this system and its model is due to

- the *number* of *interacting*, coupled, concurrent components. Complex behaviour is often a consequence of a large number of feedback loops;
- the variety of component *formalisms*. Often, a mix of software and hardware, continuous and discrete components occurs;
- the variety of *views*, at different levels of *abstraction*.

A model of a system such as the one described above may be valid (within a particular experimental context) at a certain *level of abstraction*. This level of abstraction, which may be different for each of the components, is determined by the available knowledge, the questions to be answered about the system's behaviour, the required accuracy of answers, *etc.* Orthogonal to the choice of model abstraction level is the selection of suitable *formalisms* in which the models are described. The choice of formalism is related to the abstraction level, the amount of data that can be obtained to calibrate the model, the availability of solvers/simulators for that formalism, as well as to the kind of questions which need to be answered.

3.2 Forrester System Dynamics

To fully explain the model, we now present the semantics of the Forrester System Dynamics (FSD) formalism. The FSD formalism (Cellier, 1991) describes the variation of material-like quantities or levels. The variation is determined by birth rates (*BR*) and death rates (*DR*). *BR* and *DR* are graphically represented as valves to the left and right respectively of boxes denoting the levels. Levels may influence each other by influencing each other's *BR* and *DR*.

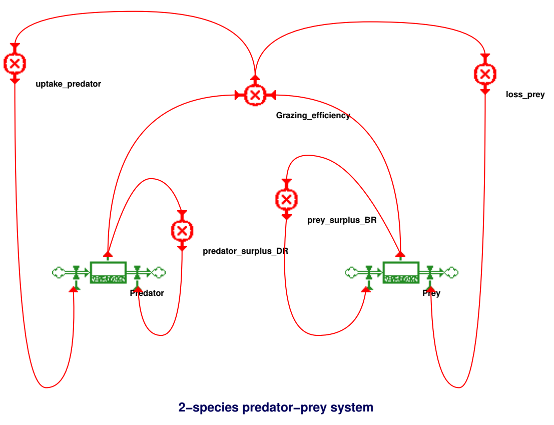

Figure 7: Predator-prey, FSD formalims.

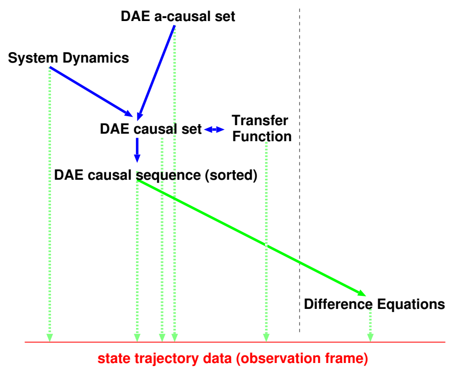

Figure 8: Formalism transformation.

The influences are given by algebraic functions. Figure 7 shows a typical interaction between a predator and a prey. The (product) interaction between predator and prey populations influences the predator's birth rate and the prey's death rate. The System Dynamics semantics is given by mapping each of the level/*BR*/*DR* combinations onto an Ordinary Differential Equation

$$\frac{d\ level}{dt} = BR - DR.$$

The operations such as product and sum are mapped onto the appropriate algebraic equations and couplings are mapped onto algebraic equalities.

Based on this relationship between the System Dynamics and the ODE formalisms, translation of any model in the first formalism to the same model described in the second formalism is possible. In practice, this implies that a compiler can translate any model in the first formalism onto a (behaviourally equivalent) model in the second formalism. In Figure 8, a small part of "formalism space" is depicted in the form of a Formalism Transformation Graph (FTG). The different formalisms described before are shown as nodes in the graph. The vertical striped line denotes the distinction between continuous models (on the left) and discrete models (on the right). The well known Difference Equations formalism is often implicitly used in numerical simulators: ODEs are discretised by means of a suitable numerical scheme and the resulting difference equations are iteratively solved. Suitable refers to the nature of the equations as well as to the accuracy requirements. The arrows denote a homomorphic relationship "can be mapped onto", implemented as a transformation between formalisms. The vertical, dotted lines denote the existence of a *solver* or *simulation kernel* which is capable of simulating a model, thus generating a trajectory. A trajectory is really a model of the system in the data formalism (time/value tuples). Obviously, the experimental frame of a trajectory is very limited. In a denotational sense, traversing the graph makes *semantics* of models in formalisms explicit: the meaning of a model/formalism is given by mapping it onto some known formalism (whose meaning may in turn be given by mapping it onto an even more basic formalism*, etc.*). Though a multi-step mapping may seem cumbersome, it can be perfectly and correctly performed by tools. The advantage of this approach is that the introduction of a *new formalism* only requires the description of the mapping onto the nearest formalism as well as the implementation of this mapping in the form of a model translator. It is often meaningful to introduce a new formalism for a specific application, encoding particular properties and constraints of the application. Often, translation involves some loss of information. This loss may be a blessing in disguise as it entails a reduction in complexity, hopefully leading to an increase in (simulation) performance. Usually, the aim of multi-step mapping is to eventually reach the trajectory level. Another major use for formalism transformation is the answering of particular questions about the system. Some questions can only be answered in the context of a particular formalism. In case of a FSD model for example, the visual inspection of the model can provide insight into influences. If the model is mapped onto a set of Algebraic and Ordinary Differential Equations, a dependency analysis may reveal algebraic cycles not apparent at the System Dynamics level. At this same level, one may check whether parts of the model are linear. If so, these parts may be solved symbolically by means of computer algebra. Also, transformation to the Laplace domain (*i.e.,* to a Transfer Function form) leads to a plethora of techniques for stability analysis. Finally, the transformation, through numerical simulation, to the data level, allows for quantitative analysis of problems posed in initial value form.

Note how the larger the number of intermediate formalisms becomes, the higher the possibility for optimization along the way will be.

Above all, the traversal described above is the basis for the meaningful coupling of models described in different formalisms. This will be discussed next.

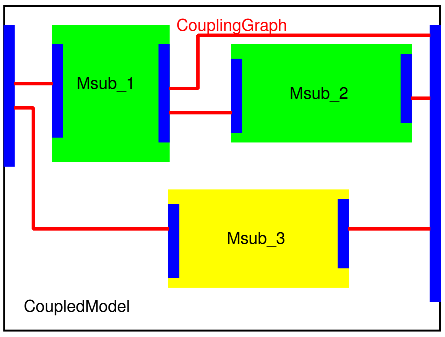

Figure 9: Multi-formalism coupled model.

3.3 Coupled Model Transformation

Figure 9 shows a multi-formalism coupled model (the different shades denote different formalisms). Models of this type are said to belong to the *network* or *coupled* formalism. When we observe a structured model at this level, we can only make meaningful assertions about its structure, its outside connections (its interface) and its components, not about its overall meaning or behaviour ! Formally, a coupled model has the form

$$CM \equiv \langle id, interface, S, C \rangle$$

The model is identified by a unique identifier *id* (for example, a name or reference). The *interface* is a set of connectors or ports to the outside. Associated with the ports are allowed values as well as causality. Meaningful causalities are $\{in, out, inout\}$. The set S contains the sub-models (or at least their unique identifiers). The coupling information is contained in a graph structure C. For non-causal, continuous models, the graph is undirected. For causal models, the graph is directed. Obviously, a coupled model is only valid if types and causalities of connected ports are compatible. In certain cases, the graph may be annotated with extra information. In case of discrete-event models, a tie-breaking function is usually required to select between simultaneous events (Zeigler, 1984a).

If all the sub-models are described in the same formalism F, it may be possible to (recursively, bottom-up, if the sub-models are coupled models in their own right) replace the coupled model (at least conceptually) by one atomic model of type F. In this case, F is called *closed under coupling* (or under composition). The closure property has to be proven for each F. The property often holds by construction. In case of Differential Algebraic Equations (DAEs), connections $\{connect(port_i, port_j)\}$ are replaced by $port_i = port_j$ coupling equations. Together with the sub-models' equations, these form a DAE. In formalisms such as Bond Graphs, information about the physical nature of variables allows one to generate either the above type of equations in case of coupling of across variables (this corresponds to Kirchoff's voltage law in electricity) and an equation summing all connected values to zero for through variables (this corresponds to Kirchoff's current law in electricity). In discrete-event models, implementing closure involves the correct time-ordered *scheduling* of sub-model events. The most imminent event will always be processed first. The tie-breaking function is used to resolve conflicts due to simultaneous events (an artifact of the high level of abstraction).

If a coupled model consists of sub-models expressed in different formalisms, different approaches are possible:

- A *super-formalism* can be used which subsumes the different formalisms of the sub-models. The different sub-models are thus described in the same formalism. The Hybrid DAE (Vangheluwe, 2000) and DEVS&DESS (Zeigler et al., 2000) formalisms integrate continous and discrete modelling constructs. Meaningful super-formalisms which truly add expressiveness and reduce complexity are rare. Bond Graphs (Cellier, 1991) are a good example of the integration of different domains (mechanical, electrical, hydraulical).
- Another approach is to *transform* the different sub-models to one *common formalism*. Which formalism to transform to depends on the questions asked. The closest common formalism for the WWTP (DAE) and fish farm (System Dynamics) example is the DAE formalism. By transforming the fish farm model to a set of DAEs, and using the closure property of the DAE formalism, it becomes possible to answer questions about the overall model. Often, the target formalism is the trajectory level.
- In the *co-simulation approach*, each of the sub-models is simulated with a formalism-specific simulator. Interaction due to coupling is resolved at the trajectory level. Compared to transformation to a common formalim before simulation, this approach, though appealing from a software engineering point of view, discards a variety of useful information. Questions can *only* be answered at the trajectory level. Furthermore, there are obvious speed and numerical accuracy problems for continuous formalisms (Foster and Yelmgren, 1997). The approach is meaningful only for discrete-event formalisms. In this realm, it is the basis of the DoD High Level Architecture (HLA) for simulator interoperability.

The transformation to a common formalism mentioned above proceeds as follows:

1. Start from a coupled multi-formalism model. Check consistency of this model (*e.g.,* whether causalites and types of connected ports match).
2. Cluster all models described in the same formalism.
3. For each cluster, implement closure under coupling.
4. Look for the “best” common formalism in the Formalism Transformation Graph all the remaining different

formalisms can be transformed to. In the worst case, this will be the trajectory level in which case the approach falls back to co-simulation. Which common formalism is best depends on a quality metric which can take into account transformation speed, potential for optimization*, etc.* .

5. Transform all the sub-models to the common formalism.

6. Implement closure under coupling of the common formalism.

Certain questions about a system can only be answered in certain formalisms, necessitating model transformation to the appropriate formalism. A side-effect of mapping onto a common formalism is the great potential for optimization of the flattened model, as well as the reduced number of (optimized) simulation kernels needed. When models are used to *communicate* knowledge between humans or tools, the formalism used must be clearly and uniquely specified. As described further on, meta-modelling allows for explicit reasoning about often subtle differences between formalisms. This is achieved by explicitly modelling both syntax and semantics of formalisms.

3.4 The Formalism Transformation Graph

To describe which formalism transformation are possible, the Formalism Transformation Graph (FTG) is used (Vangheluwe, 2000). As an example, the transformations currently known to the authors are shown in Figure 10. The line at the bottom denotes the state trajectory formalism. In principle, this should be a single node, but a line was used for aesthetic purposes. The vertical dashed line denotes the crude division between continuous and discrete(-event) models. Vertical dashed arrows denote the existence of a simulator which maps an abstract model onto a state trajectory (given initial conditions, parameters*, etc.*). It is noted that iterative simulation can be seen as a special case of model transformation.

4 META-MODELLING

A proven method to achieve flexibility for a modelling language to support many formalisms is to model the language itself. This approach is demonstrated in, among others, the *domain modelling environment* (DOME) (Honeywell, 1999; Engstrom and Kruger, 2000) and the *multigraph architecture* (MGA) (Karsai et al., 2000). To illustrate this notion, consider the state transition diagram in Figure 11 (in the Deterministic Finite State Automata formalism). When in the ON *state*, a *transition* to the OFF state occurs when the *condition* $t > 2$ becomes true and this generates an alarm *action*. The state, transition, condition, and action elements are part of any state transition diagram and their dependencies can be modelled as shown in Figure 12 (in the Entity-Relationship formalism). This model specifies a family of state transition diagrams where each instantiation has states

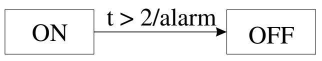

Figure 11: A state transition diagram model.

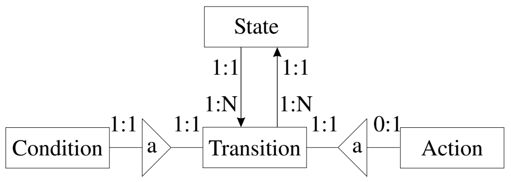

Figure 12: A model of state transition diagram models.

that are connected by transitions. Each state can have any number of outgoing transitions, indicated by the $1 : 1$ and $1 : N$ cardinality on the downward arrow in the figure, *i.e.,* each state can have between 1 and N transitions and each transition has to exit between 1 and 1 states (*i.e.,* it has to be connected to one and only one state), where N represents any number. Each transition can enter only one state and each state may have any number of entering transitions indicated by the cardinality on the upward arrow. The transitions between states have two attributes, one condition that allows the state transition to be taken and one optional (indicated by the $0 : 1$ cardinality) action. The model in Figure 12 can be used to specify different state transition diagram formalisms, *e.g.,* the action attribute can be made mandatory for each transition by changing the cardinality from $0 : 1$ to $1 : 1$. Furthermore, actions can be associated with states as well.

Such a model of the modelling language is called a *meta*-model. It prescribes the possible mathematical structures (formalisms) that can be expressed in the modelling language and can be tailored to specific needs of particular domains. From the meta-model specification, the modelling language, graphical or textual, can then be instantiated automatically. This requires the meta-model modelling formalism to be sufficiently rich and support the constructs needed to define a modelling language. To allow for easy extension, the meta-model modelling formalism can be modelled by a *meta-meta*-model. This meta-meta-model specification captures the basic elements that can be used to design a meta-model modelling formalism. In case new concepts and structures are required, these can be conveniently modelled at a meta-meta-level.

For example, the meta-model in Figure 12 is limited to the family of state transition diagrams. This restriction can be removed by modelling the model of state transition diagrams in Figure 12 by the meta-meta-model in Figure 13. It contains an abstract representation of the mechanisms that are part of the state transition diagrams meta-model, *i.e.,* entities (states, transitions), attributes (actions and conditions), and relations between them. This meta-meta-model groups entities and relations by an *object* model component and each of them optionally has any number of attributes. It also shows that each relation connects to one entity as

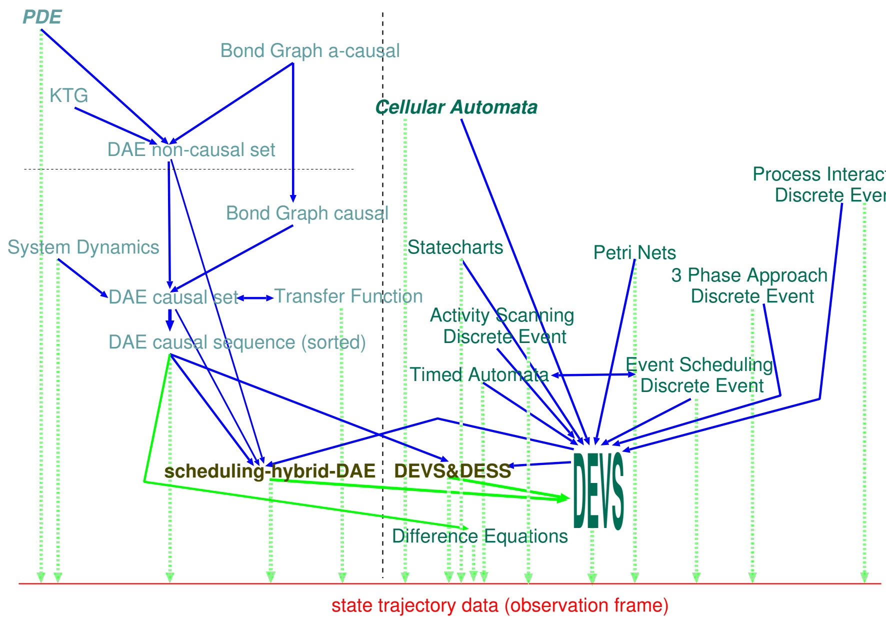

Figure 10: Formalism Transformation Graph.

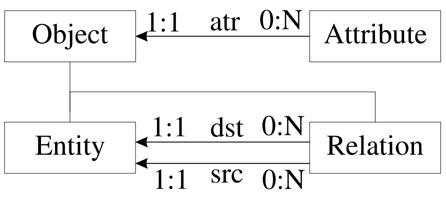

Figure 13: A state transition diagram meta-meta-model.

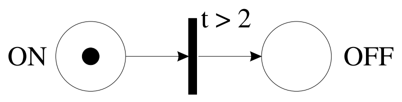

Figure 14: A Petri net model.

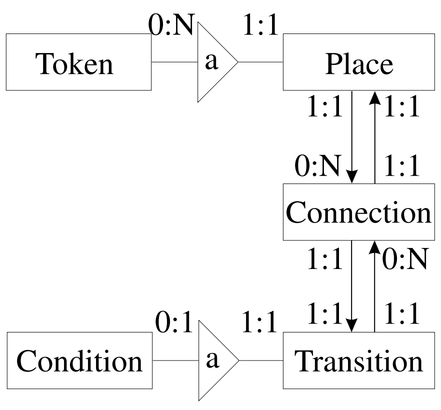

Figure 15: A model of Petri net models.

its source (marked src) and one entity as its destination (marked dst), and, therefore, directed links can be used. The meta-meta-model specification language has to consist of entities, attributes to specify the cardinality and relations to specify the three types, (i) source relations (src), (ii) destination relations (dst), and (iii) attribute relations (atr). Given the meta-meta-model in Figure 13, a broader family of meta-models can be described. For example, a Petri net as illustrated in Figure 14 consists of *places* (shown as transparent circles) and *transitions* (shown as solid rectangles). Each place has *connections* to transitions and each transition to places. A transition may have a condition and when this condition is true and all its input places, *i.e.,* places that are the source to the transition, contain a *token* (shown as a black dot inside a place), it may fire (the corresponding transition may be executed). The Petri nets meta-model shown in Figure 15 specifies places and transitions that can connect to one another. Furthermore, the tokens are specified as optional attributes of a place and conditions as an optional attribute of a transition. The family of Petri nets that can be instantiated from this meta-model allows places with multiple tokens. However, in some modelling paradigms, only one token per place is allowed, which can be conveniently changed in the meta-model by specifying $0:1$ cardinality, and, consequently, constraining the family of Petri nets. Note that this represents both a static and dynamic constraint. When the Petri net is initialized, it can be ensured by the model editor that each place has been assigned at most one token. However, it is still necessary to dynamically check whether this constraint is violated during execution. To be able to fully specify modelling formalisms, a meta-level formalism is often extended with a textual *constraint language* complementary to the often graphical base. This language can be used to rule out semantically incorrect models and greatly reduce the family

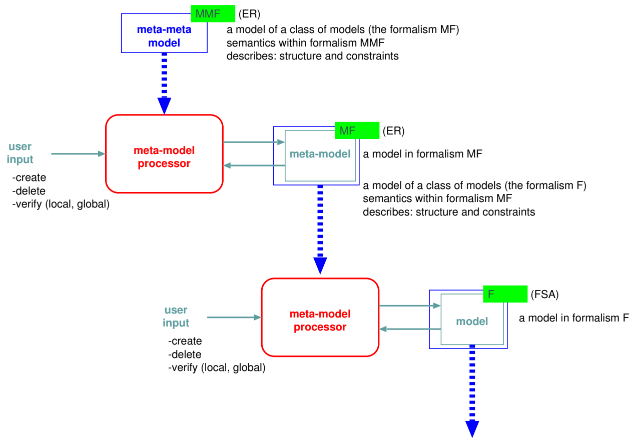

Figure 16: Meta-··· models.

of models that can be modelled (Nordstrom, 1999). For example, a model of a family of Petri nets could include the constraint that there should never be more than ten tokens in the net (to model a resources limit) by an additional specification `Token.allInstances->size < 10`.

The outlined meta-modelling approach leads to the layered structure in Table 2. The model row represents a particular model such as the state transition diagram in Figure 11. At a meta-level, the meta-model row represents a class of models (or formalism), *e.g.,* the model of the family of state transition diagrams in Figure 12. This meta-model has meaning within a formalism which is specified by a meta-meta-model. This could be the model in Figure 13.

The apparent advantage of this approach is the tremendous flexibility that can be achieved.

This is manifested in the ease with which new formalisms can be designed. By adapting the model of a modelling formalism, and automatically generating a prototype of the modelling environment, design choices can be rapidly evaluated.

Figure 16 depicts how meta-models are models too. An FSA model is constructed within the FSA formalism. In the meta-modelling approach, the FSA formalism is explicitly modelled. The FSA formalism's meta-model has meaning within the Entity-relationship formalism.

4.1 Meta-modelling Model Transformation

Upto now, only model *syntax* has been considered in meta-modelling. It is however above all necessary to describe model *semantics*. At the core of this is the specification of model transformation.

Some of the manipulations we are interested in are:

- Formalism transformation: Given a model in a certain formalism, these transformations convert it into a model, but expressed in another formalism. For Modelling and Simulation, possible transformations are found in the Formalism Transformation Graph.

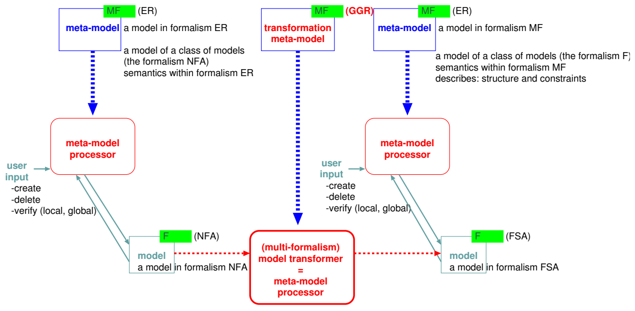

Figure 17: Model transformation meta-specification.

- Model optimization: These transformations do not change the formalism in which the model is expressed. Their application results in a reduction of model complexity.
- Code Generation: These transformations produce a textual representation of the model (subject to syntactic constraints) suitable for interpretation by a simulator.
- Simulator specification: These specifications give the operational semantics of the model.

All these tasks depend on the formalisms of interest. However, since models determined by some meta-model can always be described as graphs (subject to the constraints given by the meta-model), these tasks can be performed by a generic graph-transformation algorithm. Therefore it makes sense to combine meta-modelling and *graph-grammars* (Dorr, 1995) in a unifying framework. Meta-models determine the classes of graphs that are allowed on the LHS and RHS of a graph-grammar rule. Furthermore, the rules themselves, and the grammars, can be viewed as models in the graph-grammar (GGR) formalism, which itself is described in a meta-model. This is depicted in Figure 17. Graph Grammars are composed of rules, each mapping a graph on the left-hand side (LHS) to a graph on the right-hand side (RHS). A graph-grammar is applied to an input graph (called host graph) in order to perform a transformation. When a match is found between the LHS of a rule and a part of the host graph, this subgraph is replaced by the RHS of the rule. Rules may also have a condition that must be satisfied for the rule to be applied, as well as actions to be performed when the rule is executed. A rewriting system iteratively applies matching rules in the grammar to the graph, until no more rules are applicable.

On the one hand, the use of a model (in the form of a graph grammar) of graph transformations has some advantages over an implicit representation (embedding the transformation computation in a program) (Blonstein et al., 1996):

- it is an abstract, declarative, high level representation, and
- the theoretical foundations of graph rewriting systems may assist in proving correctness and convergence properties of the transformation tool.

Level	Description	Example
meta-meta-model	Model used to specify modelling languages	`Relation hasDestination Entity`
meta-model	Model used to specify models, an instantiation of a meta-meta-model	`State connectsTo Transition`
model	The description of an object in a specific formalism	`when t > 2 transition from ON to OFF`

Table 2: Three layer meta-modelling structure.

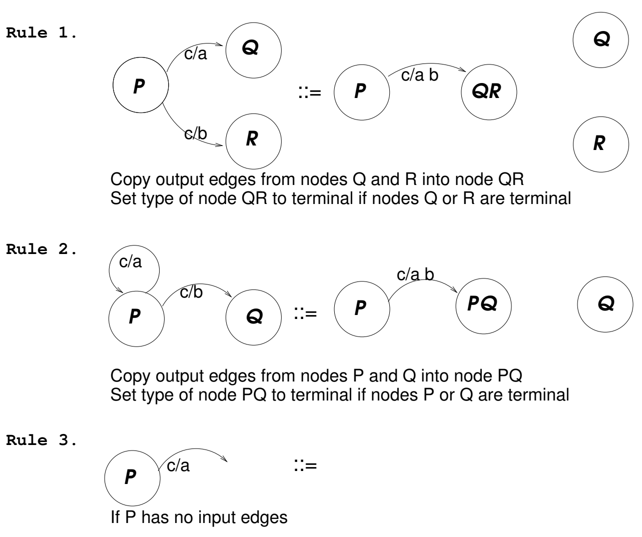
Rule 1.
P
c/a
Q
c/b
R
::=
P
c/a b
QR
Q
R
Copy output edges from nodes Q and R into node QR
Set type of node QR to terminal if nodes Q or R are terminal
Rule 2.
c/a
P
c/b
Q
::=
P
c/a b
PQ
Q
Copy output edges from nodes P and Q into node PQ
Set type of node PQ to terminal if nodes P or Q are terminal
Rule 3.
P
c/a
::=
If P has no input edges

Figure 18: NFA to DFA Graph Grammar.

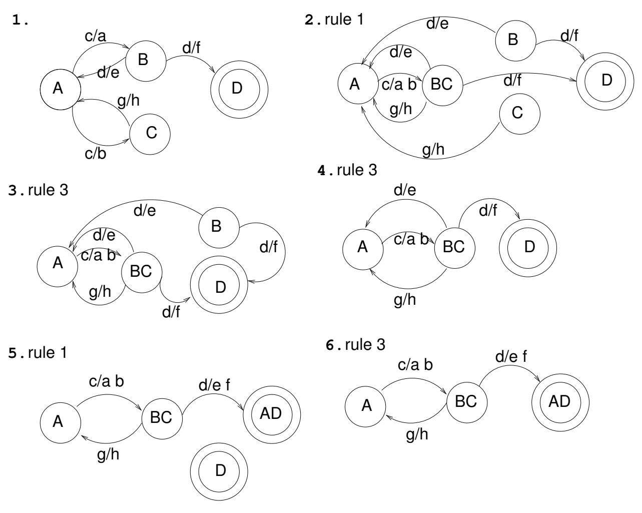

Figure 19: NFA to FSA Transformation.

On the other hand, the use of graph grammars is constrained by efficiency. In the most general case, subgraph isomorphism testing is NP-complete. However, the use of small subgraphs on the left hand side of graph grammar rules, as well as using node labels and edge labels can greatly reduce the search space.

As an example, Figure 18 gives the rules for transforming a non-deterministic finite state automaton (NFA) into an equivalent deterministic finite state automaton (DFA). Figure 19 shows the application of this graph grammar to a simple NFA.

It should be noted that the *operational semantics* of a formalism may be expressed in the form of a graph grammar model. This, by encoding the formalism's transition function. As such, graph grammar application corresponds to simulation. We can thus regard the graph grammar as an *executable specification*. This approach is desirable for its generality, since it can be applied to a wide class of formalisms. There is, however, a tradeoff made between generality and efficiency. As a general rule, customized, hand-coded, formalism-specific simulation algorithms are more efficient. The approach of relying on graph grammars is expensive due to the nature of the graph matching algorithm. However, there are other motivations, both theoretical and practical:

- Explicitly defining the operational semantics of any formalism should be part of the design of a simulator. The specification is then the basis for any implementation.
- The specification and its interpretation provide a framework (a *reference implementation*) for verifying and testing different (hand-crafted, efficient) implementations.
- It provides a portable simulator, since it is more abstract than a hand-coded implementation.
- It allows for reasoning about the described systems. For example, it allows for the definition of general algorithms for bisimulation.

5 CONCLUSIONS

We have presented a comprehensive overview of multi-paradigm Modelling and Simulation. The focus was on the concept of multi-formalism modelling of complex systems, and its relation to the semantics of models. Furthermore, meta-modelling was presented as a means of dealing with multiple formalisms. Currently, the first two authors are developing AToM^3, a meta-modelling environment. AToM^3 uses graph grammars to specify (at a meta-level), the transformations in the Formalism Transformation Graph. AToM^3 can be found at `http://moncs.cs.mcgill.ca/MSDL/research/projects/AToM3`.

ACKNOWLEDGEMENT

Prof. Vangheluwe gratefully acknowledges partial support for this work by a National Sciences and Engineering Research Council of Canada (NSERC) Individual Research Grant.

REFERENCES

Balci, O. (1988). The implementation of four conceptual frameworks for simulation modeling in high-level languages. In Abrams, M., Haigh, P., and Comfort, J., editors, *Proceedings of the 1988 Winter Simulation Conference*, pages 287–295. Society for Computer Simulation International (SCS).

Balci, O. (1997). Principles of simulation model validation, verification, and testing. *Transactions of the Society for Computer Simulation International*, 14(1):3–12. Special Issue: Principles of Simulation.

Barros, F. J., Zeigler, B. P., and Fishwick, P. A. (1998). Multimodels and dynamic structure models: an integration of DSDE/DEVS and OOPM. In Medeiros, D., Watson, E., and M.S., M., editors, *Proceedings of the 1998 Winter Simulation Conference*, pages 413–419. Society for Computer Simulation International (SCS).

Blonstein, D., Fahmy, H., and Grbavec, A. (1996). Issues in the practical use of graph rewriting.

Cellier, F. E. (1991). *Continuous System Modeling*. Springer-Verlag, New York.

Dorr, H. (1995). Efficient graph rewriting and its implementation.

Engstrom, E. and Kruger, J. (2000). A meta-modeler's job is never done: Building and evolving domain-specific tools with DOME. In Varga, A., editor, *IEEE International Symposium on Computer-Aided Control System Design*, pages 83–88. IEEE Computer Society Press. Anchorage, Alaska.

Foster, L. and Yelmgren, K. (1997). Accuracy in DoD High Level Architecture Federations. In Obaidat, M. S. and Illgen, J., editors, *Summer Computer Simulation Conference (SCSC'97)*, pages 451–460. Society for Computer Simulation International (SCS). Arlington, Virginia.

Gordon, G. (1996). *System Simulation*. Prentice Hall of India, second edition.

Honeywell (1999). DOME guide. http://www.htc.honeywell.com/dome/, Honeywell Technology Center, Honeywell. version 5.2.1.

Karsai, G., Nordstrom, G., Ledeczi, A., and Sztipanovits, J. (2000). Specifying graphical modeling systems using constraint-based metamodels. In Varga, A., editor, *IEEE International Symposium on Computer-Aided Control System Design*, pages 89–94. IEEE Computer Society Press. Anchorage, Alaska.

Magee, B. (1985). *Popper*. Fontana Press (An Imprint of HarperCollins Publishers), London.

Nance, R. E. (1981). The time and state relationships in simulation modeling. *Communications of the ACM*, 24(4):173–179.

Nordstrom, G. G. (1999). *Metamodeling – Rapid Design and Evoluion of Domain-Specific Modeling Environments*. PhD dissertation, Vanderbilt University, Electrical Engineering.

Rumbaugh, J., Jacobson, I., and Booch, G. (1999). *The Unified Modeling Language Reference Manual*. Object Technology Series. Addison-Wesley.

Vangheluwe, H. L. (2000). DEVS as a common denominator for multi-formalism hybrid systems modelling. In Varga, A., editor, *IEEE International Symposium on Computer-Aided Control System Design*, pages 129–134. IEEE Computer Society Press. Anchorage, Alaska.

Vangheluwe, H. L., Kerckhoffs, E. J., and Vansteenkiste, G. C. (2001). Computer automated modelling of complex systems. In Kerckhoffs, E. J. and Snorek, M., editors, 15th *European Simulation Multi-conference (ESM)*, pages 7–18. Society for Computer Simulation International (SCS). Prague, Czech Republic.

Vangheluwe, H. L. and Vansteenkiste, G. C. (1996). SiE: Simulation for the Future. *Simulation*, 66(5):331 – 335.

Wymore, A. W. (1967). *A Mathematical Theory of Systems Engineering – the Elements*. Wiley Series on Systems Engineering and Analysis. Wiley.

Zeigler, B. P. (1984a). *Multifacetted Modelling and Discrete Event Simulation*. Academic Press, London.

Zeigler, B. P. (1984b). *Theory of Modelling and Simulation*. Robert E. Krieger, Malabar, Florida.

Zeigler, B. P., Praehofer, H., and Kim, T. G. (2000). *Theory of Modelling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press, second edition.