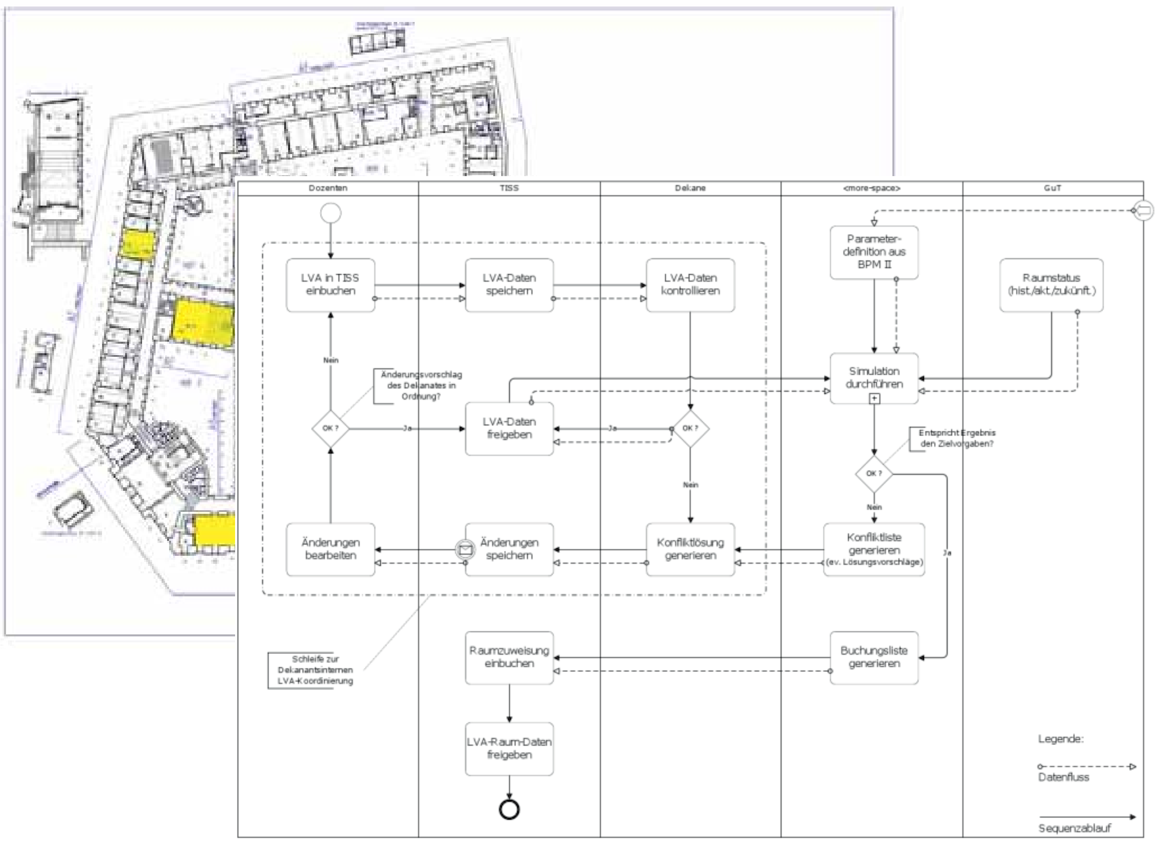
SNE SIMULATION NEWS EUROPE



Volume 19 Number 1
April 2009

doi: 10.11128/sne.19.1.0992
DOI Reprint ISSN 0929-2268

Print ISSN 2305-9974
Online ISSN 2306-0271

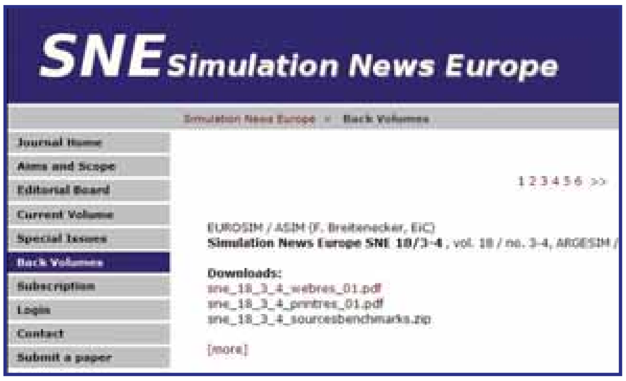
Journal on Developments and Trends in Modelling and Simulation

Membership Journal for Simulation Societies in EUROSIM

Dear Readers,

*With this SNE issue, SNE 19/1, our SNE volumes are on publication schedule again, and we start with new publication strategy and new distribution strategy for SNE: printed SNE, and electronic SNE (eSNE). Previously some back issues of SNE could be downloaded from the ARGESIM website www.argesim.org for free, and members of some EUROSIM societies had download access to SNE issues – in both cases SNE copies were available as pdf- files in web resolution (low resolution). After some negotiations with EUROSIM societies and other simulation groups, we are able to introduce generally also the electronic SNE—**eSNE** for download: web resolution eSNE in PDF format can be downloaded for free from SNE website www.sne-journal.org, print-resolution e-SNE in pdf format and all sources and additional information of the benchmark solution in zip-format can be downloaded for society subscribers.*

The picture at right shows the new SNE website, with download access for web-resolution SNE 18/3-4 (highlighted)—after login also print-resolution eSNE 18/3-4 and the zipped sources of benchmarks solutions published in SNE 18/3-4 would be available for download. The new SNE webpage is not only documenting SNE and its contents, but also it is also organising SNE administration in submission, reviewing, publishing, and distributing contributions. Authors may submit contributions of any kind via this system, followed by an appropriate reviewing. Also reports from societies will be managed by this system, so that new SNE issues can be compiled in time on basis of reviewed and refined contributions (starting end of 2009)



EUROSIM / ASIM (F. Breitenecker, EiC)
Simulation News Europe SNE 18/3-4 , vol. 18 / no. 3-4, ARGESIM /

Downloads:
sne_18_3_4_webres_01.pdf
sne_18_3_4_printres_01.pdf
sne_18_3_4_sourcesbenchmarks.zip

[more]

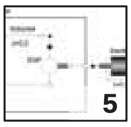
The Technical Notes in this SNE issue conclude the topic 'physical and hybrid modelling' the emphasis of the last four SNE issues: D. Broman and P. Fritzson discuss high-order acausal models, H. Nilson introduces type-based structural analysis for modular models, and F. Casella et al extend equation-based object-oriented with CACSD. A four-page solution to Benchmark C3 'Generlazed Class-E Amplifier' compares different modelling techniques – announcing a new type of Benchmark solutions (details in SNE 19/3-4). Furthermore, a Short Note on an implementation of a distributed consensus algorithm continues the series of Software Notes. The title page of this issue, an animation snapshot student movement through lecture halls at TU Vienna, announces the emphasis of the next issues: hybrid and discrete systems.

The News Sections section follows the new style introduced in SNE 18/3-4: after the EUROSIM Data & Quick Info, which summarizes the relevant basic information of all EUROSIM societies, SNE publishes actual reports from EUROSIM societies, which have sent in recent information. Furthermore SNE continues with reports and conferences of simulation societies and simulation groups: conferences series MATHMOD, Modelica, EOOLT, and MOSIM, and the SCS initiatives MISS and M&SNet. We hope, readers enjoy the novelties and the content, and we thank all contributors, members of the editorial boards, and people of our ARGESIM staff for co-operation in producing this SNE issue.

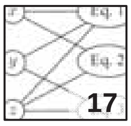
Felix Breitenecker, editor-in-chief, eic@sne-journal.org

SNE 19/1 in Three Minutes

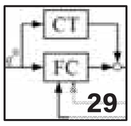
Impressum, Table of Contents: page 4


5

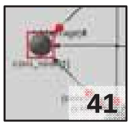
High-Order Acausal Models combines higher-order functions with acausal models to simplify the creation of reusable simulation software.


17

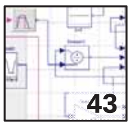
Type-Based Structural Analysis for Modular Systems of Equations attributes structural types to equation systems to speed up classification of such systems.


29

Beyond Simulation: Computer Aided Control System Design Using Equation-Based Object Oriented Modelling for the Next Decade allows a brief gaze into the future.


41

Implementation of a distributed consensus algorithm with OMNeT++ introduces a event-driven simulator generator and illustrates its use on a consensus algorithm.


43

ARGESIM Benchmark:
C3 'Analysis of a Class E Amplifier' using Dymola

16 pp.

EUROSIM short info gives a short overview over EUROSIM and its national societies, and provides contact addresses.

Reports section publishes reports from SLOSIM, SIMS, CEA-SMSG, ROMSIM, CROSSSIM, PSCS, ASIM.
It provides two ASIM conference reports.

In the *Conference series* section, MATHMOD, MOSIM, Modelica and EOOLT conferences are covered.

Simulation centers section presents two initiatives of SCS: MISS and M&SNet.

EUROSIM 2010

organised by CSSS

September 2010, Prague, Czech Republic

EUROSIM 2010

7th EUROSIM Congress on Modelling and Simulation

Eurosim Congress the most important modelling and simulation event in Europe

September 5-10, 2010, Prague, Czech Republic

Congress Venue
The Congress will take place in Prague, the capital city of Czech Republic, at the Congress Center of Masaryk College, part of Czech Technical University, in cooperation with the Faculty of Electrical Engineering of CTU.

About Czech Technical University in Prague
Czech Technical University celebrates 300 years of its history in 2007. Under the name Estate Engineering Teaching Institute in Prague was founded by the rescript of the Emperor Josef I of 18 January 1707 on the basis of a petition of Christian Josef Willenberg (1676-1731). This school was reorganized in 1806 as the Prague Polytechnic, and, after the disintegration of the former AustroHungarian Empire in 1918, transformed in to the Czech Technical University in Prague.

About EUROSIM
EUROSIM, the federation of European simulation societies, was set up in 1989. Its purpose is to promote, especially through local simulation societies, the idea of modelling and simulation in different fields, industry, research and development. At present, EUROSIM has 14 full members and 4 observer members.

Congress Scope and Topics
The Congress scope includes all aspects of continuous, discrete (event) and hybrid modelling, simulation, identification and optimisation approaches. Contributions from both technical and non-technical areas are welcome. Two basic tracks will be organized: M&S Methods and Technologies and M&S Applications.

Czech Republic - EUROSIM 2010 Host Country
The Czech Republic is a country in the centre of Europe. It is interesting for its 1,000-year-long history, rich culture and diverse nature. The country is open to new influences and opportunities thanks to a high level of industrial infrastructure, safety measures and plural media. The location of the Czech Republic in the very heart of Europe contributes to the fact that one can get there easily and fast. Usually all it takes to enter the country is a valid passport. The Czech Republic belongs to the Schengen zone. The need for a visas to enter the Czech Republic is very exceptional.

Prague - EUROSIM 2010 Host City
Prague is a magical city of bridges, cathedrals, gold-tipped towers and church spires, whose image has been mirrored in the surface of the Vltava River for more than a millennium.Walking through the city, you will quickly discover that the entire history of European architecture has left splendid representatives of various periods and styles. There are Romanesque, Gothic, Renaissance, Baroque and Classicist buildings, as well as more modern styles, such as Art Nouveau and Cubist. A poet once characterized Prague as a symphony of stones.

About CSSS
CSSS (The Czech and Slovak Simulation Society) has about 150 members in 2 groups connected to the Czech and Slovak national scientific and technical societies (Czech Society for Applied Cybernetics and Informatics, Slovak Society for Applied Cybernetics and Informatics). Since 1992 CSSS is a full member of EUROSIM.

Invitation
Czech and Slovak Simulation Society is greatly honored with the congress organisation and will do the best to organise an event with a high quality scientific programme with some other acompanied actions but also with some unforgettable social events.

Mikuláš Alexík, EUROSIM president,
Miroslav Šnorek, president of CSSS, EUROSIM 2010 Chair

Table of Contents

SNE 19(1) Reprint doi: 10.11128/sne.19.1.0992

SNE Contact. SNE Editors / ARGESIM
c/o Inst. f. Analysis and Scientific Computation
Vienna University of Technology
Wiedner Hauptstrasse 8-10, 1040 Vienna, AUSTRIA
Tel + 43 - 1- 58801-10115 or 11455, Fax – 42098
office@sne-journal.org; www.sne-journal.org

SNE Simulation News Europe ISSN 1015-8685 (0929-2268).

Scope: Technical Notes and Short Notes on developments in modelling and simulation in various areas (application and theory) and on bechmarks for modelling and simulation, membership information for EUROSIM and Simulation Societies.

Editor-in-Chief: Felix Breitenecker, Inst. f. Analysis and Scientific Computing, Vienna University of Technology, Wiedner Hauptstrasse 8-10, 1040 Vienna, Austria; Felix.Breitenecker@tuwien.ac.at

Layout: Markus Wallerberger, markus.wallerberger@gmx.at

Administration, Web: Anna Mathe, anna.mathe@tuwien.ac.at

Printed by: Grafisches Zentrum, TU Vienna, Wiedner Hauptstrasse 8-10, 1040, Vienna, Austria

Publisher: ARGESIM/ASIM; c/o Inst. for Scientific Computing, TU Vienna, Wiedner Hauptstrasse 8-10, 1040 Vienna, Austria, and ASIM (German Simulation Society), c/o Wohlfartstr. 21b, 80939 Munchen © ARGESIM/ASIM 2009

High-Order Acausal Models

David Broman, Peter Fritzson, Linköping University, Sweden, *{davbr, petfr}@ida.liu.se*

SNE Simulation Notes Europe SNE 19(1), 2009, 5-16, doi: 10.11128/sne.19.tn.09921

Current equation-based object-oriented (EOO) languages typically contain a number of fairly complex language constructs for enabling reuse of models. However, support for model transformation is still often limited to scripting solutions provided by tool implementations. In this paper we investigate the possibility of combining the well known concept of higher-order functions, used in standard functional programming languages, with acausal models. This concept, called Higher-Order Acausal Models (HOAMs), simplifies the creation of reusable model libraries and model transformations within the modeling language itself. These transformations include general model composition and recursion operations and do not require data representation/reification of models as in metaprogramming/metamodeling. Examples within the electrical and mechanical domain are given using a small research language. However, the language concept is not limited to a particular language, and could in the future be incorporated into existing commercially available EOO languages.

Introduction

Modeling and simulation have been an important application area for several successful programming languages, e.g., Simula [6] and C++ [24]. These languages and other general-purpose languages can be used efficiently for discrete time/event-based simulation, but for continuous-time simulation, other specialized tools such as Simulink [15] are commonly used in industry. The latter supports causal block-oriented modeling, where each block has defined input(s) and output(s). However, during the past two decades, a new kind of language has emerged, where differential algebraic equations (DAEs) can describe the continuous-time behavior of a system. Moreover, such languages often support hybrid DAEs for modeling combined continuous-time and discrete-time behavior.

These languages enable modeling of complex physical systems by combining different domains, such as electrical, mechanical, and hydraulic. Examples of such languages are Modelica [10, 17], Omola [1], gPROMS [3, 20], VHDLAMS [5], and χ (Chi) [13, 27].

A fundamental construct in most of these languages is the *acausal model*. Such a model can encapsulate and compose both continuous-time behavior in form of DAEs and/or other interconnected sub-models, where the direction of information flow between the sub-models is not specified. Several of these languages (e.g., Modelica and Omola) support object-oriented concepts that enable the composition and reuse of a-causal models. However, the possibilities to perform *transformations* on models and to create generic and reusable transformation libraries are still usually limited to tool-dependent scripting approaches [7, 11, 26], despite recent development of metamodeling/metaprogramming approaches like MetaModelica [12].

In functional programming languages, such as Haskell [23] and Standard ML [16], standard libraries have for a long time been highly reusable, due to the basic property of having functions as first-class values. This property, also called *higher-order functions*, means that functions can be passed around in the language as any other value.

In this paper, we investigate the combination of acausal models with higher-order functions. We call this concept *Higher-order Acausal Models (HOAMs)*.

A similar idea called *first-class relations on signals* has been outlined in the context of functional hybrid modeling (FHM)[18]. However, the work is still at an early stage and it does not yet exist any published description of the semantics. By contrast, our previous work's main objective has been to define a formal operational semantics for a subset of a typical EOO language [4]. From the technical results of our earlier work, we have extracted the more general ideas of HOAM, which are presented in this paper in a more informal setting.

An objective of this paper is to be accessible both to engineers with little functional language programming background, as well as to computer scientists with minimal knowledge of physical acausal modeling. Hence, the paper is structured in the following way to reflect both the broad intended audience, as well as presenting the contribution of the concept of HOAMs:

- The fundamental ideas of traditional higher-order functions are explained using simple examples. Moreover, we give the basic concepts of acausalmodels when used for modeling and simulation (Section 1).
- We state a definition of higher order acausal models (HOAMs) and outline motivating examples. Surprisingly, this concept has not been

widely explored in the context of EOO-languages (Section 1).

- The paper gives an informal introduction to physical modeling in our small research language called Modeling Kernel Language (MKL) (Section 2).
- We give several concrete examples within the electrical and mechanical domain, showing how HOAMs can be used to create highly reusable modeling and model transformation/composition libraries (Section 3).

Finally, we discuss future perspectives of higher-order acausal modeling (Section 4), and related work (Section 5).

1 The basic idea of high-order

In the following section we first introduce the well established concept of anonymous functions and the main ideas of traditional higher-order functions. In the last part of the section we introduce acausal models and the idea of treating models with acausal connections to be higher-order.

1.1 Anonymous functions

In functional languages, such as Haskell [23] and Standard ML [16], the most fundamental language construct is functions. Functions correspond to partial mathematical functions, i.e., a function $f : A \rightarrow B$ gives a mapping from (a subset of) the domain A to the codomain B.

In this paper we describe the concepts of higher-order functions and models using a tiny untyped research language called *Modeling Kernel Language (MKL)*. The language has similar modeling capabilities as parts of the Modelica language, but is primarily aimed at investigating novel language concepts, rather than being a full-fledged modeling and simulation language.

In this paper an informal example-based presentation is given. However, a formal operational semantics of the dynamic elaboration semantics for this language is available in [4]. In MKL, similar to general purpose functional languages, functions can be defined to be *anonymous*, i.e., the function is defined without an explicit naming. For example, the expression

```
func(x) {x*x}
```

is an anonymous function that has a formal parameter `x` as input parameter and returns `x` squared. Formal parameters are written within parentheses after the `func` keyword, and the expression representing the body of the function is given within curly parentheses; in this case `{x*x}`. An anonymous function can be applied by writing the function before the argument(s) in a parenthesized list, e.g. `(3)`:

```
func(x){x*x}(3)
→ 3*3
→ 9
```

The lines starting with a left arrow (→) show the evaluation steps when the expression is executed. However, it is often convenient to name values. Since anonymous functions are treated as values, they can be defined to have a name using the `def` construct in the same way as constants.

```
def pi = 3.14
def power2 = func(x){x*x}
```

Here, both `pi` and function `power2` can be used within the defined scope. Hence, the definitions can be used to create new expressions for evaluation, for example:

```
power2(pi)
→ power2(3.14)
→ 3.14 * 3.14
→ 9.8596
```

1.2 Higher-order functions

In many situations, it is useful to pass a function as an argument to another function, or to return a function as a result of executing a function. When functions are treated as values and can be passed around freely as any other value, they are said to be *first-class citizens*. In such a case, the language supports *higher-order functions*.

Definition 1 (Higher-order function)
A higher-order function is a function that

1. takes another function as argument, and/or
2. returs a function as the result

Let us first show the former case where functions are passed as values. Consider the following function definition of `twice`, which applies the function `f` two times on `y`, and then returns the result.

```
def twice = func(f,y){
    f(f(y))
};
```

The function `twice` can then be used with an arbitrary function `f`, assuming that types match. For example, using it in combination with `power2`, this function is applied twice.

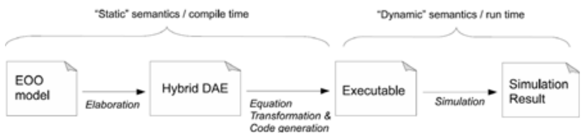

"Static" semantics / compile time
"Dynamic" semantics / run time
EOO model
Elaboration
Hybrid DAE
Equation Transformation & Code generation
Executable
Simulation
Simulation Result

Figure 1. Outline of typical compilation and simulation process for an EOO language tool.

```
twice(power2,3)
→ power2(power2(3))
→ power2(3*3)
→ power2(9)
→ 9*9
→ 81
```

Since `twice` can take any function as an argument, we can apply `twice` to an anonymous function, passed directly as an argument to the function `twice`.

```
twice(func(x){2*x-3},5)
→ func(x){2*x-3}(func(x){2*x-3}(5))
→ func(x){2*x-3}(2*5-3)
→ func(x){2*x-3}(7)
→ 2*7-3
→ 11
```

Let us now consider the second part of Definition 1, i.e., a function that returns another function as the result.

In mathematics, functional composition is normally expressed using the infix operator $\circ$. Two functions $f : X \to Y$ and $g : Y \to Z$ can be composed to $g \circ f$: $X \to Z$, by using the definition $(g \circ f)(x) = g(f(x))$. The very same definition can be expressed in a language supporting higher-order functions:

```
def compose = func(g,f) {
    func(x){ g(f(x)) }
};
```

This example illustrates the creation of a new anonymous function and returning it from the `compose` function. The function composes the two functions given as parameters to `compose`. Hence, this example illustrates both that higher-order functions can be applied to functions passed as arguments (using formal parameters `f` and `g`), and that new functions can be created and returned as results (the anonymous function). To illustrate an evaluation trace of the composition function, we first define another function `add7`.

```
def add7 = func(x){7+x};
```

and then compose `power2` and `add7` together, forming a new function `foo`:

```
def foo = compose(power2, add7);
→ def foo = func(x){power2(add7(x))};
```

Note how the function `compose` applied to `power2` and `add7` evaluates to an anonymous function. Now, the new function `foo` can be applied to some argument, e.g.

```
foo(4)
→ func(x){power2(add7(x))}(4)
→ power2(add7(4))
→ power2(7+4)
→ power2(11)
→ 11*11
→ 121
```

The simple numerical examples given here only show the very basic principle of higher-order functions. In functional programming other more advanced usages, such as list manipulation using functions `map` and `fold`, are very common.

1.3 Elaboration and simulation of acausal models

In conventional object-oriented programming languages, such as Java or C++, the behavior of classes is described using methods. On the contrary, in equation-based object-oriented languages, the continuous-time behavior is typically described using differential algebraic equations and the discrete-time behavior using constructs generating events. This behavior is grouped into abstractions called classes or models (Modelica) or entities and architectures (VHDL-AMS). From now on we refer to such an abstraction simply as *models*.

Models are blue-prints for creating *model instances* (in Modelica called components). The models typically have well-defined interfaces consisting of ports (also called connectors), which can be connected together using *connections*. A typical property of EOO-languages is that these connections usually are acausal, meaning that the direction of information flow between model instances is not defined at modeling time.

In the context of EOO languages, we define acausal (also called non-causal) models as follows:

Definition 2 (Acausal model)
An acausal model is an abstraction that encapsulates and composes

1. continuous-time behavior in form of differential algebraic equations (DAEs).
2. other interconnected acausal models, where the direction of information flow between submodels is not specified.

In many EOO languages, acausal models also contain conditional constructs for handling discrete events. Moreover, connections between model instances can typically both express potential connections (across) and flow (also called through) connections generating sum-to-zero equations. Examples of acausal models in both MKL and Modelica are given in Figure 2 and described in Section 2.1.

A typical implementation of an EOO language, when used for modeling and simulation, is outlined in Figure 1. In the first phase, a hierarchically composed acausal model is *elaborated* (also called flattened or instantiated) into a hybrid DAE, describing both continuous-time behavior (DAEs) and discrete-time behavior (e.g., when-equations). The second phase performs *equation transformations and code generation*, which produces executable target code. When this code is executed, the actual simulation of the model takes place, which produces a simulation result. In the most common implementations, e.g., Dymola [7] or OpenModelica [26], the first two phases occur during compile time and the simulation can be viewed as the run-time. However, this is not a necessary requirement of EOO languages in general, especially not if the language supports structurally dynamic systems (e.g., Sol [29], FHM [18], or MosiLab [8]).

1.4 Higher-order acausal models

In EOO languages models are typically treated as compile time entities, which are translated into hybrid DAEs during the elaboration phase. We have previously seen how functions can be turned into first-class citizens, passed around, and dynamically created during evaluation. Can the same concept of higher-order semantics be generalized to also apply to acausal models in EOO languages? If so, does this give any improved expressive power in such generalized EOO language?

In the next section we describe concrete examples of acausal modeling using MKL. However, let us first define what we actually mean by higher-order acausal models.

Defintion 3 (Higher-order acausal model (HOAM))
A higher-order acausal model is an acausal model, which can be

1. parametrized with other HOAMs.
2. recursively composed to generate new HOAMs.
3. passed as argument to, or returned as result from functions.

In the first case of the definition, models can be parametrized by other models. For example, the constructor of a automobile model can take as argument another model representing a gearbox. Hence, different automobile instances can be created with different gearboxes, as long as the gearboxes respect the interface (i.e., type) of the gearbox parameter of the automobile model. Moreover, an automobile model does not necessarily need to be instantiated with a specific gearbox, but only *specialized* with a specific gearbox model, thus generating a new more specific model.

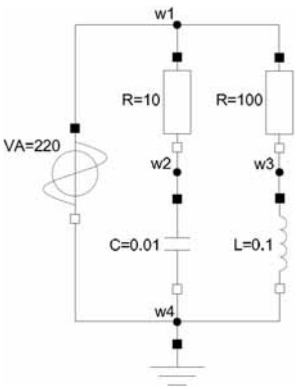
The second case of Definition 3 states that a model can reference itself; resulting in a recursive model definition. This capability can for example express models composed of many similar parts, e.g., discretization of flexible shafts in mechanical systems or pipes in fluid models.

Finally, the third case emphasizes the fact that HOAMs are first-class citizens, e.g., that models can be both passed as arguments to functions and created and returned as results from functions. Hence, in the same way as in the case of higher-order functions, generic reusable functions can be created that perform various tasks on arbitrary models, as long as they respect the defined types (interfaces) of themodels' formal parameters. Consequently, this property enables *model transformations* to be defined and executed within the modeling language itself. For example, certain discretizations of models can be implemented as a generic function and stored in a standard library, and then reused with different user defined models.

Some special and complex language constructs in currently available EOO languages express part of the described functionality (e.g., the redeclare and for-equation constructs in Modelica). However, in the next sections we show that the concept of acausal higher-order models is a small, but very powerful and expressive language construct that subsumes and/or can be used to define several other more complex language constructs. If the end user finds this more functional approach of modeling easy or hard depends of course on many factors, e.g., previous programming language experiences, syntax preferences, and mathematical skills. However, from a semantic point of view, we show that the approach is very expressive, since few language constructs enable rich modeling capabilities in a relatively small kernel language.

2 Basic physical modeling in MKL

To concretely demonstrate the power of HOAMs, we use our tiny research language Modeling Kernel Language (MKL). The higher-order function concept of the language was briefly introduced in the previous



```
def Circuit = model(){
  def w1 = Wire();
  def w2 = Wire();
  def w3 = Wire();
  def w4 = Wire();
  Resistor(w1,w2,10);
  Capacitor(w2,w4,0.01);
  Resistor(w1,w3,100);
  Inductor(w3,w4,0.1);
  VSourceAC(w1,w4,220);
  Ground(w4);
};
```

```
model Circuit
  Resistor R1(R=10);
  Capacitor C(C=0.01);
  Resistor R2(R=100);
  Inductor L(L=0.1);
  VsourceAC AC(VA=220);
  Ground G;
  equation
   connect(AC.p, R1.p);
   connect(R1.n, C.p);
   connect(C.n, AC.n);
   connect(R1.p, R2.p);
   connect(R2.n, L.p);
   connect(L.n, C.n);
   connect(AC.n, G.p);
end Circuit;
```

Figure 2. Model of a simple electrical circuit. (a) graphical model of the circuit, (b) corresponding MKL model definition, (c) Modelica model of the same circuit.

section. In this section we informally outline the basic idea of physical modeling in MKL; a prerequisite for Section 3, which introduces higher-order acausal models using MKL.

2.1 A simple electrical circuit

To illustrate the basic modeling capabilities of MKL, the classic simple electrical circuit model is given in Figure 2. Part (a) shows the graphical layout of the model, (b) shows the corresponding textual model given in MKL. For clarity to the readers familiar with the Modelica language, we also compare with the same model given as Modelica textual code (c).

In MKL, models are always defined anonymously. In the same way as for anonymous functions, an anonymous model can also be given a name, which is in this example done by giving the model the name `circuit`. The model takes zero formal parameters, given by the empty tuple (parenthesized list) to the right of the keyword model. The contents of the model is given within curly braces. The first four statements define four new *wires*, i.e., connection points from which the different components (model instances) can be connected. The six components defined in this circuit correspond to the layout given in Figure 2a. Consider the first resistor instantiated using the following:

```
Resistor(w1, w2, 10);
```

The two first arguments state that wires `w1` and `w2` are connected to this resistor. The last argument expresses that the resistance for this instance is 10 Ohm. Wire `w2` is also given as argument to the capacitor, stating that the first resistor and the capacitor are connected using wire `w2`. Modeling using MKL differs in several ways compared to Modelica (Figure 2c). First, models are not defined anonymously in Modelica and are not treated as firstclass citizens. Second, the way acausal connections are defined between model instances differs. In MKL, the connection (in this electrical case a wire), is created and then connected to the model instances by giving it as arguments to the creation of sub-model instances. In Modelica, a special `connect`-equation construct is defined in the language. This construct is used to define binary connections between connectors of sub-model instances. From a user point of view, both approaches can be used to express acausal connections between model instances. Hence, we let it be up to the reader to judge what the most natural way of defining interconnections is. However, from a formal semantics point of view, in regards to HOAMs, we have found it easier to encode connections using ordinary parameter passing style.

2.2 Connections, variables and flow nodes

The concept of wire is not built into the language. Instead, it is defined using an anonymous function, referring to the built-in constructs `var()` and `flow()`:

```
def Wire = func(){
     (var(),flow())
};
```

Here, a function called `Wire` is defined by using the anonymous function construct `func`. The definition states that the function has an empty formal parameter list (i.e., takes an empty tuple `()` as argument) and returns a tuple `(var(),flow())`, consisting of two elements. A tuple is expressed as a sequence of terms separated by commas and enclosed in parentheses.

The first element of the defined tuple expresses the creation of a new unknown continuous-time variable using the syntax `var()`. The variable could also been assigned an initial value, which is used as a start value when solving the differential equation system. For example, creating a variable with initial value 10 can be written using the expression `var(10)`. Variables defined using `var()` correspond to *potential* variables, i.e., the voltage in this example.

The second part of the tuple expresses the current in the wire by using the construct `flow()`, which creates a new flow-node. This construct is the essential part in the formal semantics of [4]. However, in this informal introduction, we just accept that Kirchhoff's current law with sum to zero at nodes is managed in a correct way.

In the circuit definition (Figure 2b) we used the syntax `Wire()`, which means that the function is invoked without arguments. The function call returns the tuple `(var(),flow())`. Hence, the `Wire` definition is used for encapsulating the tuple, allowing the definition to be reused without the need to restate its definition over and over again.

2.3 Models and equations systems

The main model in this example is already given as the `Circuit` model. This model contains instances of other models, such as the `Resistor`. These models are also defined using model definitions. Consider the following two models:

```
def TwoPin = model((pv,pi),(nv,ni),v){
    v = pv - nv;
    0 = pi + ni;
};

def Resistor = model(p,n,R){
    def (_,pi) = p;
    def v = var();
    TwoPin(p,n,v);
    R*pi=v;
};
```

In the same way as for `Circuit`, these sub-models are defined anonymously using the keyword `model` followed by a formal parameter and the model's content stated within curly braces. A formal parameter can be a pattern and *pattern matching* is used for decomposing arguments. Inside the body of the model, definitions, components, and equations can be stated in any order within the same scope.

The general model `TwoPin` is used for defining common behavior of a model with two connection points. `TwoPin` is defined using an anonymous model, which here takes one formal parameter. This parameter specifies that the argument must be a 3-tuple with the specified structure, where `pv`, `pi`, `nv`, `ni`, and `v` are pattern variables. Here `pv` means positive voltage, and `ni` negative current. Since the illustrated language is untyped, illegal patterns are not discovered until run-time.

Both models contain new definitions and equations. The equation `v = pv - nv;` in `TwoPin` states the voltage drop over a component that is an instance of `TwoPin`. The definition of the voltage `v` is given as a formal parameter to `TwoPin`. Note that the direction of the causality of this formal parameter is not defined at modeling time.

The resistor is defined in a similar manner, where the third element `R` of the input parameter is the resistance. The first line `def (_,pi) = p;` is an alternative way of pattern matching where the current `pi` is extracted from `p`. The pattern `_` states that the matched value is ignored. The second row defines a new variable `v` for the voltage. This variable is used both as an argument to the instantiation of `TwoPin` and as part of the equation `R*pi=v;` stating Ohm's law. Note that the wires `p` and `n` are connected directly to the `TwoPin` instance.

The inductormodel is defined similarly to the `Resistor` model:

```
def Inductor = model(p,n,L){
    def (_,pi) = p;
    def v = var(0);
    TwoPin(p,n,v);
    L*der(pi) = v;
};
```

The main difference to the `Resistor` model is that the `Inductor` model contains a differential equation `L*der(pi) = v;`, where the `pi` variable is differentiated with respect to time using the built-in `der` operator. The other sub-models shown in this example (`Ground`, `VSourceAC`, and `Capacitor`) is defined in a similar manner as the one above.

2.4 Executing the model

Recall Figure 1, which outlined the compilation and simulation process for a typical EOO language. When a model is evaluated (executed) in MKL, this means the process of elaborating a model into a DAE. Hence, the steps of equation transformation, code generation, and simulation are not part of the currently defined language semantics. These latter steps can be conducted in a similar manner as for an ordinary Modelica implementation. Alternatively, the resulting equation system can be used for other purposes, such as optimization [14]. In the next section we illustrate several examples of how HOAMs can be used. Consequently, these examples concern the use of HOAMs during the elaboration phase, and not during the simulation phase. Further discussion on future aspects of HOAMs during these latter phases is given in Section 4.

3 Examples of higher-order modeling

In Definition 3 (Section 1.4) we defined the meaning of HOAMs, giving three statements on how HOAMs can be used. This section is divided into sub-sections, where we exemplify these three kinds of usage by giving examples in MKL.

3.1 Parameterization

A common goal of model design is to make model libraries extensible and reusable. A natural requirement is to be able to parameterize models with other models, i.e., to reuse a model by replacing some of the submodels with other models. To illustrate the main idea of parameterized acausal models, consider the following oversimplified example of an automobile model, where we use `Connection()` with the same meaning as the previous `Wire()`:

```
def Automobile = model(Engine, Tire){
    def c1 = Connection();
    def c2 = Connection();
    Engine(c1);
    Gearbox(c1,c2);
    Tire(c2); Tire(c2); Tire(c2); Tire(c2)
};
```

In the example, the automobile is defined to have two formal parameters; an `Engine` model and a `Tire` model. To create a model instance of the automobile, the model can be applied to a specific engine, e.g., a model `EngineV6` and some type of tire, e.g. `TireTypeA`:

```
Automobile(EngineV6,TireTypeA);
```

If later on a new engine was developed, e.g., `EngineV8`, a new automobile model instance can be created by changing the arguments when the model instance is created, e.g.,

```
Automobile(EngineV8,TireTypeA);
```

Hence, new model instances can be created without the need to modify the definition of the `Automobile` model. This is analogous to a higher-order function which takes a function as a parameter.

In the example above, the definition of `Automobile` was not parametrized on the `Gearbox` model. Hence, the `Gearbox` definition must be given in the lexical scope of the `Automobile` definition. However, this model could of course also be defined as a parameter to `Automobile`.

This way of reusing acausal models has obvious strengths, and it is therefore not surprising that constructs with similar capabilities are available in some EOO languages, e.g., the special `redeclare` construct in Modelica. However, instead of creating a special language construct for this kind of reuse, we believe that HOAMs can give simpler and a more uniform semantics of a EOO language.

3.2 Recursively defined models

In many applications it is enough to hierarchically compose models by explicitly defining model instances within each other (e.g., the simple `Circuit` example). However, sometimes several hundreds of model instances of the same model should be connected to each other. This can of course be achieved manually by creating hundreds of explicit instances. However, these results in very large models that are hard to maintain and get an overview of.

One solution could be to add a loop-construct to the EOO language. This is the approach taken in Modelica, with the `for`-equation construct. However, such an extra language construct is actually not needed to model this behavior. Analogously to defining recursive functions, we can define *recursive models*. This gives the same modeling possibilities as adding the `for`-construct. However, it is more declarative and we have also found it easier to define a compact formal semantics of the language using this construct.

Consider Figure 3 which shows a Mechatronic model, i.e., a model containing components from both the electrical and mechanical domain. The left hand side of the model shows a simple direct current (DC) motor. The electromotoric force (EMF) component converts electrical energy to mechanical rotational energy. If we recall from Section 1, the connection between electrical components was defined using the `Wire` definition. However, in the rotational mechanical domain, the connection is instead defined by using the angle for the potential variable and the torque for flow. The rotational connection is defined as follows:

```
def RotCon = func(){(var(),flow())};
```

In the middle of the model in Figure 3 a rotational body with Inertia $J = 0.2$ is defined. This body is connected to a flexible shaft, i.e., a shaft which is divided into a number of small bodies connected in series with a spring and a damper in parallel in between each pair of bodies. N is the number of shaft elements that the shaft consists of. A model of the mechatronic system is described by the following MKL source code.

```
def MechSys = model(){
    def c1 = RotCon();
    def c2 = RotCon();
```

```
        DCMotor(c1);
        Inertia(c1,c2,0.2);
        FlexibleShaft(c2,RotCon(),120);
    };
```

The most interesting part is the definition of the component `FlexibleShaft`. This shaft is connected to the Inertia to the left. To the right, an empty rotational connection is created using the construction `RotCon()`, resulting in the right side not being connected. The third argument states that the shaft should consist of 120 elements.

Can these 120 elements be describedwithout the need of code duplication? Yes, by the simple but powerful mechanism of recursively defined models. Consider the following self-explanatory definitions of `ShaftElement`:

```
def ShaftElement = model(ca,cb){
    def c1 = RotCon();
    Spring(ca,c1,8);
    Damper(ca,c1,1.5);
    Inertia(c1,cb,0.03);
};
```

This model represents just one of the 120 elements connected in series in the flexible shaft. The actual flexible shaft model is recursively defined and makes use of the `ShaftElement` model:

```
defrec FlexibleShaft = model(ca,cb,n){
   if(n==1)
      ShaftElement(ca,cb)
   else{
      def c1 = RotCon();
      ShaftElement(ca,c1);
      FlexibleShaft(c1,cb,n-1);
   };

};
```

The recursive definition is analogous to a standard recursively defined function, where the `if`-expression evaluates to false, as long as the count parameter `n` is not equal to 1. For each recursive step, a new connection is created by defining `c1`, which connects the shaft elements in series. Note that the last element of the shaft is connected to the second port of the `FlexibleShaft` model, since the shaft element created when the `if`-expression is evaluated to true takes parameter `cb` as an argument.

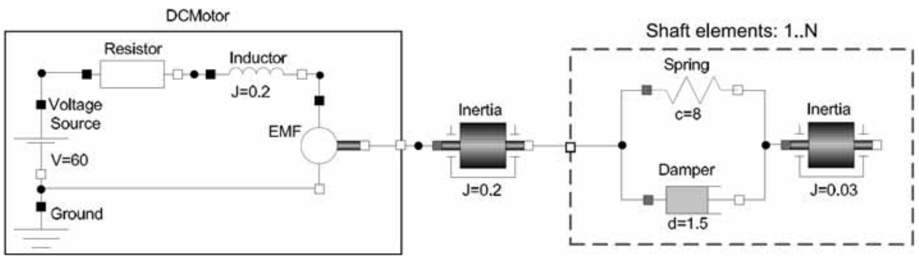


Figure 3. A mechatronic system with a direct current (DC) motor to the left and a flexible shaft to the right. The flexible shaft consists of 1 to N elements, where each element includes an inertia, a spring, and a damper.

When the `MechSys` model is elaborated using our MKL prototype implementation, it results in a DAE consisting of 3159 equations and the same number of unknowns. It is obviously beneficial to be able to define recursive models in cases such as the one above, instead of manually creating 120 instances of a shaft element.

However, it is still a bit annoying to be forced to write the recursive model definition each time one wants to serialize a number of model instances. Is it possible to capture and define this serialization behavior once and for all, and then reuse this functionality?

3.3 Higher-order functions for generic model transformation

In the previous section we have seen how models can be reused by applying models to other models, or to recursively define models. In this section we show that it is indeed possible to define several kinds of *model transformations* by using higher-order functions. These functions can in turn be part of a modeling language's standard library, enabling reuse of model transformation functions.

Recall the example from Section 1.2 of higher-order functions returning other anonymously defined functions. Assume that we want to create a generic function, which can take any two models that have two ports defined (`Resistor`, `Capacitor`, `ShaftElement` etc.), and then compose them together by connecting them in parallel, and then return this new model:

```
def composeparallel = func(M1,M2){
   model(p,n){
      M1(p,n);
      M2(p,n);
   }
};
```

However, our model `Resistor` etc. does not take two arguments, but 3, where the last one is the value for the particular component (resistance for the `Resis-`

tor, inductance for the Inductor etc.). Hence, it is convenient to define a function that sets the value of this kind of model and returns a more *specialized* model:

```
def set = func(M,val){
   model(p,n){
      M(p,n,val);
   }
};
```

For example, a new model Foo that composes two other models can be defined as follows:

```
def Foo = composeparallel(set(Resistor, 100),
                          set(Inductor, 0.1));
```

A standard library can then further be enhanced with other generic functions, e.g., a function that composes two models in series:

```
def composeserial = func(M1,M2,Con){
   model(p,n){
      def w = Con();
      M1(p,w);
      M2(w,n);
   }
};
```

However, this time the function takes a third argument, namely a connector, which is used to create the connection between the models created in series. Since different domains have different kinds of connections (Wires, RotCon etc.), this must be supplied as an argument to the function. These connections are defined as higher-order functions and can therefore easily be passed as a value to the composeserial function.

We have now created two simple generic functions which compose models in parallel and in series. However, can we create a generic function that takes a model M, a connector C, and an integer n, and then returns a new model where n number of models M has been connected in series, using connector C? If this is possible, we do not have to create a special recursive model for the FlexibleShaft, as shown in the previous section.

Fortunately, this is indeed possible by combining a generic recursive model and a higher-order function. First, we define a recursive model recmodel:

```
defrec recmodel = model(M,C,ca,cb,n){
   if(n==1)
      M(ca,cb)
   else{
      def c1 = C();
      M(ca,c1);
      recmodel(M,C,c1,cb,n-1);
   };
};
```

Note the similarities to the recursively defined model FlexibleShaft. However, in this version an arbitrary model M is composed in series, using connector parameter C. To make this model useful, we encapsulate it in a higher order function, which takes a model M, a connector C, and an integer number n of the number of wanted models in series as input:

```
def serialize = func(M,C,n){
   model(ca,cb){
      recmodel(M,C,ca,cb,n);
   }
};
```

Now, we can once again define the mechatronic system given in Figure 3, but this time by using the generic function serialize:

```
def MekSys2 = model(){
   def c1 = RotCon();
   def c2 = RotCon();
   DCMotor(c1);
   Inertia(c1,c2,0.2);
   def FlexibleShaft =
         serialize(ShaftElement,RotCon,120);
   FlexibleShaft(c2,RotCon());
};
```

Even if the serialize function might seem a bit complicated to define, the good news is that such functions usually are created by library developers and not end-users. Fortunately, the end-user only has to call the serialize function and then use the newly created model. For example, to create a new model, where 50 resistors are composed in series is as easy as the following:

```
def Res50=serialize(set(Resistor,100),Wire,50)
```

4 Future perspectives of higher-order modeling

Our current design of higher-order acausal modeling capabilities as presented here is restricted to executing during the compiler (or interpreter) model elaboration phase, i.e., it cannot interact with run-time objects during simulation. However, removing this restriction gives interesting possibilities for run-time higher-order acausal modeling:

- The run-time results of simulation can be used in conjunction with models as first-class objects in the language, i.e., run-time creation of models, composition of models, and returning models. This is also useful in applications such as model-based optimization or modelbased control, influenced by results from (on-line) simulation of models, e.g., [9].

- Structural variability [8, 18, 19, 29] of models and systems of equations means that the model structure can change at run-time, e.g., change in causality and/or number of equations. Run-time support for higher-order acausal model can be seen as a general approach to structurally variable systems. These ideas are discussed in [18, 19] in the context of Functional Hybrid Modeling (FHM).

These run-time modeling facilities provide more flexibility and expressive power but also give rise to several research challenges that need to be addressed:

- How can static strong type checking be preserved?
- How can high performance from compile-time optimizations be preserved? One example is index reduction, which requires symbolic manipulation of equations.
- How can we define a formal sound semantics for such a language?

Another future generalization of higher-order acausal modeling would be to allowmodels to be propagated along connections. For example, a water source could be connected to a generic flow connection structure with unspecified media. The selection of a media of type water in the source would automatically propagate to other objects.

5 Related work

The main emphasis of this work is to explore the language concept of HOAMs in the context of EOO languages. In the following we briefly discuss three aspects of work which is related to this topic.

5.1 Functional Hybrid Modeling

As mentioned in the introduction, our notation of HOAMs has similarities to *first-class relations on signals*, as outlined in the context of Functional Hybrid Modeling (FHM) [18, 19]. The concepts in FHM are a generalization of Functional Reactive Programming (FRP) [28], which is based on reactive programming with causal hybrid modeling capabilities. Both FHM and FRP are based on *signals* that conceptually are functions over time. While FRP supports causal modeling, the aim of FHM is to support acausal modeling with structurally dynamic systems. However, the work of FHM is currently at an early stage and no published formal semantics or implementation currently exist.

HOAMs are similar to FHM's relations on signals in the sense that they are both first-class and that they can recursively reference themselves. In this paper we have showed how recursion can be used to define large structures of connected models, while in [19] ideas are outlined how it can be used for structurally dynamic systems.

One difference is that FHM's relations on signals are as its name states only relations on signals, while MKL acausal models can be parameterized on any type, e.g., other HOAMs or constants. By contrast, FHM's relation on signals can be parameterized by other relations or constants using ordinary functional abstraction, i.e., free variables inside a relation can be bound by a surrounding function abstraction. There are obvious syntactic differences, but the more specific semantic differences are currently hard to compare, since there are no public semantic specification available for any FHM language.

The work with MKL has currently focused on formalizing a kernel language for the elaboration process of typical EOO languages, such as Modelica. Hence, the formal semantics of MKL defined in [4] investigates the complications when HOAMs are combined with flow variables, generating sum-to-zero equations. How this kind of issue is handled in FHM is currently not published.

5.2 Metaprogramming and metamodeling

The notion of higher-order models is related to, but different frommetamodeling andmetaprogramming.Ametaprogram is a programthat takes other programs/models as data and produces programs/models as data, i.e., meta-programs can manipulate *object programs* [21]. A metamodel may also have a subset of this functionality, i.e., it may specify the structure of other models represented as data, but not necessarily be executable and produce other models. Staged metaprogramming can be achieved by quoting/unquoting operations applied in two or more stages, e.g., as in MetaML [25] and Template Haskell [22].

We have earlier developed a simple metaprogramming facility for Modelica by introducing quoting/unquoting mechanisms [2], but with limited ability to perform operations on code. A later extension [12] introduced general metaprogramming operations based on pattern-matching and transformations of abstract-syntax tree representations of models/programs similar to those found in many functional programming languages.

By contrast, the notion of higher-order models in this paper allows direct access to models in the language, e.g., passing models to models and functions, returning models, etc, without first representing (or view-ing, reifying) models as data. This allows more integrated access to such facilities within the language including integration with the type system. Moreover, it often implies simpler usage and increased re-use compared to what is typically offered by metaprogramming approaches.

Metaprogramming, on the other hand, offers the possibility of greater generality on the allowed operations on models, e.g., symbolic differentiation of model equations, and the possibility of compile-time only approaches without any run-time penalty.

We should also mention the common usage of interpretive scripting languages, e.g., Python, or add-on interpretive scripting facilities using algorithmic parts of the modeling language itself such as in OpenModelica [12] and Dymola [7]. This works in practice, but is less well integrated and typically a bit ad hoc. This either requires two languages (e.g., Python and Modelica), or a separate interpretive implementation of a subset of the same language (e.g., Modelica scripting) which often give some differences in semantics, ad hoc restrictions, and inconsistent or partially missing integration with a general type system.

5.3 Modelica `redeclare` and `for` equations

Modelica [17] provides a powerful facility called redeclaration, which has some capabilities of higher order models. Using redeclare, models can be passed as arguments to models (but not to functions using ordinary argument passing mechanisms e.g., at runtime), and returned from models in the context of defining a new model. For example:

```
model RefinedResistorCircuit =
 GenericResistorCircuit(
  redeclare model ResistorModel=TempResistor);
```

Redeclaration can also be used to adapt a model when it is inherited:

```
extends GenericResistorCircuit
  (redeclare model ResistorModel=TempResistor)
```

Redeclare is a compile-time facility which operates during the model elaboration phase. Moreover, using redeclare it is not possible to pass a model to a function, or to return a model from a function. Redeclaration is similar to C++ templates and Java Generics in that it allows passing types/models, but ismore closely integrated in the language since it part of the class/model concept rather than being a completely separate feature. The Modelica redeclare can be seen as a special case of the more general concept of higher-order acausal models.

Modelica also provides the concept of for-equations to express repetitive equations and connection structures. Since iteration can be expressed as recursion, also for models as shown in Section 3.2, the concept of for-equations can be expressed as a special case of the more general concept of recursive models included in higher-order acausal models.

Even though EOO languages, such as Modelica, does not support HOAMs at the syntax level, HOAMs can still be very useful as a semantic concept for describing a precise formal semantics of the language. Language constructs, such as `for`-equations, can then be transformed down to a smaller kernel language. Having a small precisely defined language semantics can then make the language specification less ambiguous, enable better formal model checking possibilities, as well as providing more accurate model exchange.

6 Conclusions

We have in this paper informally presented how the concept of higher-order functions can be combined with acausal models. This concept, which we call higher-order acausal models (HOAMs), has been shown to be a fairly simple and yet powerful construct, which enables both parameterized models and recursively defined models. Moreover, by combining it with functions, we have briefly shown how it can be used to create reusable model transformation functions, which typically can be part of a model language's standard library. The examples and the implementation were given in a small research language called Modeling Kernel Language (MKL), and it was illustrated how HOAMs can be used during the elaboration phase. However, the concept is not limited to the elaboration phase, and we believe that future research in the area of HOAMs at runtime can enable both more declarative expressiveness as well as simplified semantics of EOO languages.

Acknowledgements

We would like to thank Jeremy Siek and the anonymous reviewers for many useful comments on this paper. This research work was funded by CUGS (the National Graduate School in Computer Science, Sweden) and by Vinnova under the NETPROG Safe and Secure Modeling and Simulation on the GRID project.

References

[1] M. Andersson. *Object-Oriented Modeling and Simulation of Hybrid Systems*. PhD thesis, Dept. Automatic Control, Lund Institute of Technology, Sweden, December 1994.

[2] P. Aronsson, P. Fritzson, L. Saldamli, P. Bunus, K. Nyström. *Meta Programming and Function Overloading in OpenModelica*. In Proc. 3rd Int. Modelica Conference, pages 431–440, Linköping, Sweden, 2003.

[3] P.I. Barton. *The Modelling and Simulation of Combined Discrete/Continuous Processes*. PhD thesis, Dept. Chemical Engineering, Imperial Collage of Science, Technology and Medicine, London, UK, 1992.

[4] D. Broman. *Flow Lambda Calculus for Declarative Physical Connection Semantics*. Technical Reports in Computer and Information Science No. 1, LIU Electronic Press, 2007.

[5] E. Christen, K. Bakalar. *VHDL-AMS - A Hardware Description Language for Analog and MixedSignal Applications*. IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, 46(10):1263–1272, 1999.

[6] O.-J. Dahl, K. Nygaard. *SIMULA: an ALGOL-based simulation language*. Communications of the ACM, 9(9):671–678, 1966.

[7] Dynasim. *Dymola − Dynamic Modeling Laboratory (Dynasim AB)*. http://www.dynasim.se/ [Last accessed: April 30, 2008].

[8] C. Nytsch-Geusen et. al. *MOSILAB: Development of a Modelica based generic simulation tool supporting model structural dynamics*. In Proc. 4th Int. Modelica Conference, Hamburg, Germany, 2005.

[9] R. Franke, M. Rode, K. Krüger. *On-line Optimization of Drum Boiler Startup*. In Proc. 3rd Int. Modelica Conference, pp. 287–296, Linköping, Sweden, 2003.

[10] P. Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Press, New York, USA, 2004.

[11] P. Fritzson, P. Aronsson, A. Pop, H. Lundvall, K. Nyström, L. Saldamli, D. Broman, A. Sandholm. *OpenModelica - A Free Open-Source Environment for System Modeling, Simulation, and Teaching*. In IEEE Int. Symposium on Computer-Aided Control Systems Design, Munich, Germany, 2006.

[12] P. Fritzson, A. Pop, P. Aronsson. *Towards Comprehensive Meta-Modeling and Meta-Programming Capabilities in Modelica*. In Proc. 4th Int. Modelica Conference, pages 519–525, 2005.

[13] G. Fábián. *A Language and Simulator for Hybrid Systems*. PhD thesis, Inst. for Programming research and Algorithmics, Technische Universiteit Eindhoven, Netherlands, 1999.

[14] J. Åkesson. *Languages and Tools for Optimization of Large-Scale Systems*. PhD thesis, Dept. Automatic Control, Lund Institute of Technology, Sweden, November 2007.

[15] MathWorks. *The Mathworks − Simulink − Simulation and Model-Based Design*. http://www.mathworks.com/products/simulink/ [Last accessed: Nov. 8, 2007].

[16] R. Milner, M. Tofte, R. Harper, D. MacQuee. *The Definition of Standard ML - Revised*. The MIT Press, 1997.

[17] Modelica Association. *Modelica − A Unified Object-Oriented Language for Physical Systems Modeling − Language Specification Version 3.0*, 2007. Available from: http://www.modelica.org.

[18] H. Nilsson, J. Peterson, P. Hudak. *Functional Hybrid Modeling*. In Practical Aspects of Declarative Languages: 5th Int. Symposium, PADL 2003, vol. 2562 of LNCS, pages 376–390, New Orleans, Lousiana, USA, January 2003. Springer-Verlag.

[19] H. Nilsson, J. Peterson, P. Hudak. *Functional Hybrid Modeling from an Object-Oriented Perspective*. In Proc. 1st Int. Workshop on Equation-Based Object-Oriented Languages and Tools, pages 71–87, Berlin, Germany, 2007. Linköping University Electronic Press.

[20] M. Oh, C.C. Pantelides. *A modelling and Simulation Language for Combined Lumped and Distributed Parameter Systems*. Computers and Chemical Engineering, 20(6–7):611–633, 1996.

[21] T. Sheard. *Accomplishments and research challenges in meta-programming*. In Proc. Workshop on Semantics, Applications, and Implementation of Program Generation, vol 2196 of LNCS, pages 2–44. SpringerVerlag, 2001.

[22] T. Sheard, S.P. Jones. *Template metaprogramming for Haskell*. In Haskell '02: Proc. 2002 ACM SIGPLAN workshop on Haskell, pages 1–16, New York, USA, 2002. ACM Press.

[23] S.P. Jones. *Haskell 98 Language and Libraries – The Revised Report*. Cambridge University Press, 2003.

[24] B. Stroustrup. *A history of C++ 1979–1991*. In HOPLII: The second ACM SIGPLAN conference on History of programming languages, pages 271–297, New York, USA, 1993. ACM Press.

[25] W. Taha, T. Sheard. *MetaML and multi-stage programming with explicit annotations*. Theoretical Computer Science, 248(1–2):211–242, 2000.

[26] The OpenModelica Project. www.openmodelica.org [Last accessed: May 8, 2008].

[27] D.A. van Beek, K.L. Man, MA. Reniers, J.e. Rooda, R.R.H Schiffelers. *Syntax and consistent equation semantics of hybrid Chi*. The Journal of Logic and Algebraic Programming, 68:129–210, 2006.

[28] Zhanyong W., P. Hudak. *Functional reactive programming from first principles*. In PLDI '00: Proc. ACM SIGPLAN 2000 conference on Programming language design and implementation, pages 242–252, New York, USA, 2000. ACM Press.

[29] D. Zimmer. *Enhancing Modelica towards variable structure systems*. In Proc. 1st Int. Workshop on Equation-Based Object-Oriented Languages and Tools, pages 61–70, Berlin, Germany, 2007. Linköping University Electronic Press.

Corresponding author: David Broman,
Department of Information and Computer Science,
Linköping University, Sweden
davbr@ida.liu.se

Accepted EOOLT 2008, June 2008
Received: September 20, 2008
Accepted: December 11, 2008

Type-Based Structural Analysis for Modular Systems of Equations

Henrik Nilsson, University of Nottingham, UK, *nhn@cs.nott.ac.uk*

SNE Simulation Notes Europe SNE 19(1), 2009, 17-28, doi: 10.11128/sne.19.tn.09923

This paper investigates a novel approach to a type system for modular systems of equations; i.e., equation systems constructed by composition of individual equation system fragments. The purpose of the type system is to ensure, to the extent possible, that the composed system is solvable. The central idea is to attribute a *structural type* to equation system fragments that reflects which variables occur in which equations. In many instances, this allows over- and underdetermined system fragments to be identified separately, without first having to assemble all fragments into a complete system of equations. The setting of the paper is equation-based, non-causal modelling, specifically Functional Hybrid Modelling (FHM). However, the central ideas are not tied to FHM, but should be applicable to equation-based modelling languages in general, like Modelica, as well as to applications featuring modular systems of equations outside the field of modelling and simulation.

Introduction

An important question in the context of equation-based modelling is whether or not the system of equations describing the modelled entity is solvable. In general, this can only be answered by studying the complete system of equations, and often not even then, except by attempting to solve the equations through simulation.

This is problematic. Models are usually modular, i.e. described by combining small systems of equations into larger ones. Being able to detect problems with individual parts or their combinations without first having to put together a complete system model is generally desirable. Moreover, a system may be *structurally dynamic*, meaning that the system of equations describing its behaviour *changes* over time. This implies that the question of the solvability cannot be addressed prior to simulation.

However, establishing that a system of equations *definitely is not solvable* can be almost as helpful. Fortunately there are criteria necessary (but not sufficient) for solvability that can be checked more easily and that are applicable to model fragments. A simple example is that the number of variables (unknowns) and equations must agree. For example, Modelica as of version 3.0 [12] enforces this constraint for model fragments (and thus for a model as a whole) so as to enable early detection of common modelling mistakes. Keeping track of the variable and equation balance is also the idea behind the structural constraint delta type system [2] with similar aims.

This paper is a preliminary investigation into an improved type-based (and thus compile-time) analysis for determining when (fragments of) systems of equations *cannot* be solved. The goal is to provide improved precision compared with just counting variables and equations by attributing a *structural type* to systems of equations reflecting which variables occur in which equations. A type-based approach is adopted as that is a natural way of ensuring that model fragments can be checked in isolation. This is particularly important for structurally dynamic systems where parts of the system change over time. However, as long as the *types* of the parts remain unchanged, and are reasonably informative, a meaningful analysis can still be carried out statically, at compile-time.

The development is carried out in the context of Functional Hybrid Modelling (FHM) [14, 15], as this provides a small and manageable modelling language framework that helps keeping the focus on the essence of the problem. FHM itself is still in an early stage of development. However, the central ideas put forward in this paper are not tied to FHM, but should be applicable to equation-based modelling languages like Modelica in general, as well as to applications featuring modular systems of equations outside the field of modelling and simulation. In effect, FHM is mainly used as a convenient and concise notation for modular systems of equations.

The rest of the paper is organised as follows. Section 1 provides general background and discusses related work. Section 2 provides an overview of FHM in the interest of making this paper relatively self-contained. Section 3 then develops the idea of structural types for modular systems of equations. As an example, this is applied to a simple electrical circuit in Section 4. Finally, Section 5 discusses future work and Section 6 gives conclusions.

1 Background and related work

Object-oriented modelling languages like Modelica [12] allow models to be developed in a *modular* fashion: systems of equations describing individual components are composed into larger systems of equations describing aggregates of components, and ultimately into a complete model of the system under consideration. As with software in general, such modularity is key to addressing the complexity of large-scale development as it allows large problems to be broken down into smaller ones that can be addressed independently, enables reuse, etc.

Of course, it is possible that mistakes are made during the development of a model. If so, it is desirable to catch such mistakes early. In a modular setting, this means checking whether a component in isolation is inherently faulty, and whether two or more components are being composed appropriately. As a result, mistakes can be localised effectively, meaning it becomes a lot easier to find and correct them. In contrast, mistakes that only become evident once a system has been fully assembled are usually a lot harder to pinpoint as the symptom in itself often is not enough to suggest any particular part of the system as the root of the problem. Even more problematic is a situation where problems only reveal themselves in use, as this means the system is unreliable.

A good way to catch errors early is to employ the notion of *types*. An entity has some particular type if it satisfies the properties implied by that type. A *type system* then governs under which conditions typed entities may be combined, and determines what properties the combined entity satisfies, i.e. its type. As a simple example, consider the type `Integer`. If an entity has type `Integer`, this means that this entity satisfies the property of being an integer. Moreover, a rule of the type system would establish that *any* two entities satisfying the property of being integers can be combined using arithmetic addition into a new entity that also is an integer. This example is trivial, but as we will see, it is possible to capture much more complex properties through suitably defined types.

An important aspect of a type system is that it works solely on the basis of the *types* of the combined entities, without referring to any specific entity *instances*. This makes it possible to establish various properties of a combined entity before knowing exactly what all its parts are. This in turn allows for all manner of useful parametrisations, systems with dynamically evolving structure, etc.

This paper is concerned with equation systems properties for establishing whether a system can be solved or not. One necessary but not sufficient condition for solvability is the variable and equation balance: globally, the number of variables to solve for and the number of equations must be equal. Languages like Modelica naturally enforce this. Since version 3.0 [12], Modelica has adopted the even stricter criterion that (in essence) variables and equations must be *locally* balanced, i.e. balanced on a per component basis. Thus, in a sense, the property of being balanced is implicitly part of the type of a component in Modelica 3.0, as all well-typed components are balanced. Naturally, if all components of a model are locally balanced, this implies that the model is globally balanced.

Of course, a locally imbalanced model might still be globally balanced. To allow such models (without deferring *all* checking until a model has been fully assembled), it is necessary to *explicitly* make the variable and equation imbalance part of the type of a component. This was suggested by Nilsson *et al.* [14] and, independently, by Broman *et. al.* [2], who developed the idea in detail by integrating the notion of a "structural constraint delta" into the types of components.

Unfortunately, ensuring that the number of variables and equations agree only gives relatively weak assurances. As a simple example, consider the following system of equations, where f, g, and h are known functions, and x, y, and z are variables:

$$\begin{aligned} f(x,y,z) &= 0 \\ g(z) &= 0 \\ h(z) &= 0 \end{aligned}$$

The number of equations and variables agree. Yet it is clear that we cannot hope to solve this system of equations: x and y occur only in one equation, but we need two equations to have a chance to determine both of them. Moreover, z occurs alone in two of the equations, meaning that it may be impossible to find a value of z that satisfies them both. What we have in this case is an *underdetermined* system of equations for x and y (one equation, two variables), and an *overdetermined* system of equations for z (two equations, one variable). Note that it was possible to establish the unsolvability of this system by just considering its *structure*: which variables occur in which equations. This can be formalised through the notion of a *structurally singular* system of equations:

Definition 1 (Structually singular system of eqs.)
A system of equations is *structurally singular* iff it is not possible to put the variables and equations in a one-to one correspondence such that each variable occurs in the equation it is related to.

We now simply observe that a system of equations that is structurally singular is unsolvable.

Languages like Modelica ensure that models are not structurally singular as simulation is not possible if this is the case. However, in Modelica, this check is not carried out on a per component basis, but only once the system has been fully expanded into a "flat" system of equations. To the best of this author's knowledge, this is also the case for all similar languages. As a result, if it turns out that the final model is structurally singular, it can be very difficult to find out what the origin of the problem is.

To help overcome this difficulty, Bunus and Fritzson proposed a method to help localising the cause of any structural singularity [3, 4]. Their idea is to view the system of equations as a bipartite graph where the variables constitute one set of nodes, the equations the other set of nodes, and there is an edge between a variable and an equation if the former occurs in the latter. See Figure 1(a) and 1(b). They then use the Dulmage and Mendelsohn canonical decomposition algorithm [6] to partition the flat system of equations into three parts: one overdetermined, one underdetermined, and one where the variables and equations match up. This information is then used to help diagnose the problem and suggest remedies.

Still, it would be an advantage if mistakes that *inevitably* are going to lead to structural singularities can be flagged up early, without first having to fully expand a model. This is true in particular for structurally dynamic systems: since the system of equations describing the behaviour of the system change over time, there is no one fully expanded system in this case. This is the kind of systems we ultimately hope to address in the context of our work on Functional Hybrid Modelling [14, 15].

This paper investigates an approach to early detection of structural singularities. The basic idea is to attribute types to components such that these types characterise the *structure* of the underlying system of equations used to represent a component, or more precicely, the structure of the equations that constitute its *interface*. We refer to this as the *structural type* of the component. The fundamental idea is similar to the structural constraint delta approach suggested by Broman *et al.*. However, the structural type is much richer: instead of a single number reflecting the variable and equation imbalance, the structural type details which variables occur in which equations. That is, the structural type is essentially a bipartite graph as in the work by Bunus and Fritzson, or it can be viewed as an *incidence matrix*: see Figure 1(c).We will freely switch between these two points of view in the following.

It turns out, though, that it often will be necessary to approximate the information on which variables occur in which equations. Thus the approach of this paper is not a complete alternative to error diagnosis on the final, flat system of equations as suggested by Bunus and Fritzson, but rather complementary to it.

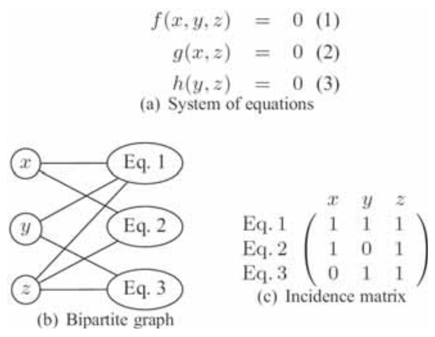


Figure 1. A system of equations and its corresponding structural representations.

2 Functional hybrid modelling

Functional Hybrid Modelling (FHM) [14, 15] is a generalisation of the central ideas of *Functional Reactive Programming* (FRP) [18]. In FRP, a functional programming language is extended with constructs for reactive programming and *causal*, hybrid, modelling, specifically *signals* (time-varying values) and functions on signals. This has proved to yield a very flexible and expressive framework for many different kinds of reactive and modelling applications [13, 9, 5, 8]. The FHM approach is similar, but *relations on signals* are added to address *non-causal* modelling.

The salient features of FRP and FHM relevant for this paper are covered in the rest of this section. The ideas are illustrated with a simple circuit example. This example is also used later in this paper. Note that FHM is currently being developed: no complete implementation exists yet. However, as explained earlier, it provides a convenient setting for this work.

2.1 Fundamental concepts

FRP is a conceptual framework. A number of concrete implementations exists. Here, we will briefly consider Yampa [13], which is most closely related to FHM. Yampa is based on two central concepts: signals and signal functions. A signal is a function from time to a value; conceptually:

$$\mathit{Signal}\ \alpha \approx \mathit{Time} \rightarrow \alpha$$

(The conceptual nature of this definition is indicated by $\approx$. $\rightarrow$ is the infix type constructor for function types.) Time is continuous, and is represented as a non-negative real number. The type parameter α

specifies the type of values carried by the signal. For example, the type of a varying electrical voltage might be *Signal Voltage*.

A *signal function* is a function from *Signal* to *Signal*:

$$SF\ \alpha\ \beta\ \approx\ Signal\ \alpha\ \rightarrow\ Signal\ \beta$$

When a value of type $SF\ \alpha\ \beta$ is applied to an input signal of type *Signal* α, it produces an output signal of type *Signal* β. Signal functions are *first class entities* in Yampa. Signals, however, are not: they only exist indirectly through the notion of signal function. Additionally, signal functions satisfies a causality requirement: at any point in time, the output must not depend on future input (this is *temporal* causality, a notion distinct from the notion of causality in "non-causal modelling.")

The output of a signal function at time t is uniquely determined by the input signal on the interval $[0,t]$. If a signal function is such that the output at time t only depends on the input at the very same time instant t, it is called *stateless*. Otherwise it is *stateful*.

2.2 First-class signal relations

A natural mathematical description of a continuous signal function is that of an ODE in explicit form. A function is just a special case of the more general concept of a *relation*. While functions usually are given a causal interpretation, relations are inherently non-causal. Differential Algebraic Equations (DAEs), which are at the heart of non-causal modelling, express dependences among signals without imposing a causality on the signals in the relation. Thus it is natural to view the meaning of a DAE as a non-causal *signal relation*, just as the meaning of an ODE in explicit form can be seen as a causal signal function. Since signal functions and signal relations are closely connected, this view offers a clean way of integrating non-causal modelling into an Yampa-like setting.

Similarly to the signal function type SF of Yampa (Section 2.1), the type $SR\ \alpha$ stands for a relation on a signal of type α. Like signal functions, signal relations are first class entities, as will become clear in the following. Specific relations use a more refined type; e.g., for the derivative relation **der** we have the typing:

$$\mathbf{der} :: SR\ (\ Real, Real\ )$$

Since a signal carrying pairs is isomorphic to a pair of signals, we can understand **der** as a binary relation on two real-valued signals. Signal relations are constructed as follows:

sigrel *pattern* **where** *equations*

The pattern introduces *signal variables* that at each point in time are bound to the *instantaneous* value of the corresponding signal. Given a pattern p of type t, $p :: t$, we have:

sigrel p **where** … $:: SR\ t$

Consequently, the equations express relationships between instantaneous signal values. This resembles the standard notation for differential equations in mathematics. For example, consider $x' = f(y)$, which means that the instantaneous value of the derivative of (the signal) x at every time instant is equal to the value obtained by applying the function f to the instantaneous value of y.

There are two styles of basic equations:

$$e_1 = e_2$$
$$sr \lozenge e_3$$

where e_i are expressions (possibly introducing new signal variables), and sr is an *expression* denoting a signal relation. We require equations to be well-typed. Given $e_i :: t_i$, this is the case iff $t_1 = t_2$ and $sr :: t_3$.

The first kind of equation requires the values of the two expressions to be equal at all points in time. For example:

$$f\ x\ =\ g\ y$$

where f and g are ordinary, pure functions (we follow standard functional programming practice and denote ordinary function application simply by juxtapositioning, without any parentheses.)

The second kind allows an arbitrary relation to be used to enforce a relationship between signals. The symbol $\lozenge$ can be thought of as *relation application*; the result is a constraint which must hold at all times. The first kind of equation is just a special case of the second in that it can be seen as the application of the identity relation. Thus, with I denoting the identity relation, an equation $e_1 = e_2$ could also be written $I \lozenge (e_1, e_2)$. For another example, consider a differential equation like $x' = f(x)$. Using the notation above, this equation can be written:

$$\mathbf{der} \lozenge (x, f\,x\,y)$$

where **der** is the relation relating a signal to its derivative. For notational convenience, we will often use a notation closer to standard mathematical practice:

$$\mathbf{der}\ x = f\,x\,y$$

The meaning is exactly as in the first version. Thus, in the second form, **der** is *not* a pure function operat-

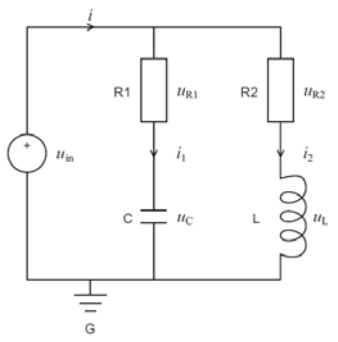


Figure 2. A simple electrical circuit.

ing only on instantaneous signal values. It is a (stateful) signal function operating on the underlying signal.

We illustrate the ideas above by modelling the electrical circuit in Figure 2 (adapted from [11]). The type *Pin* is a record type describing an electrical connection. It has fields *v* for voltage and *i* for current.

```
twoPin :: SR (Pin, Pin, Voltage)
twoPin = sigrel (p, n, u) where
    u = p.v − n.v
    p.i + n.i = 0
resistor :: Resistance → SR (Pin, Pin)
resistor r = sigrel (p, n) where
    twoPin ◊ (p, n, u)
    r * p.i = u
inductor :: Inductance → SR (Pin, Pin)
inductor l = sigrel (p, n) where
    twoPin ◊ (p, n, u)
    l * der p.i = u
capacitor :: Capacitance → (Pin, Pin)
capacitor c = sigrel (p, n) where
    twoPin ◊ (p, n, u)
    c * der u = p.i
```

The resistor, inductor and capacitor models are defined as extensions of the *twoPin* model. This is accomplished using functional abstraction rather than any Modelica-like class concept. Note how parameterized models are defined through functions *returning* relations, e.g. *resistor*. Since the parameters (like *r* of *resistor*) are normal function arguments, *not* signal variables, their values remain unchanged throughout the lifetime of the returned relations (in Modelica terms, they are parameter variables). As signal relations are first class entities, signal relations can be parameterized on other signal relations in the same way.

To assemble these components into the full model, a Modelica-inspired **connect**-notation is used as a convenient abbreviation for connection equations. In FHM, this is just syntactic sugar that is expanded to basic equations: equality constraints for connected potential quantities and a sum-to-zero equation for connected flow quantities. In the following, connect is only applied to *Pin* records, where the voltage field is declared as a potential quantity whereas the current field is declared as a flow quantity.

We assume that a voltage source model *vSourceAC* and a ground model *ground* are available in addition to the component models defined above. Moreover, we are only interested in the total current through the circuit, and, as there are no inputs, the model thus becomes a *unary* relation:

```
simpleCircuit :: SR Current
simpleCircuit = sigrel i where
    resistor 1000 ◊ (r1p, r1n)
    resistor 2200 ◊ (r2p, r2n)
    capacitor 0.00047 ◊ (cp, cn)
    inductor 0.01 ◊ (lp, ln)
    vSourceAC 12 ◊ (acp, acn)
    ground ◊ gp
connect acp r1p r2p
connect r1n cp
connect r2n lp
connect acn cn ln gp
i = r1p.i + r2p.i
```

There is no need to declare variables like *r1p*, *r1n*: their types are inferred. Note the signal relation expressions like *resistor* 1000 to the left of the signal relation application operators ◊.

As an illustration of signal relation application, let us expand resistor 1000 ◊ (*r1p*, *r1n*) using the definitions of twoPin and *resistor*. The result is is the following three equations, where u1 is a fresh variable:

```
u1 = r1p.n − r1n.v
r1p.i + r1n.i = 0
1000 * r1p.i = u1
```

2.3 Dynamic structure

Yampa can express highly structurally dynamic systems. Ultimately, we hope to integrate as much of that functionality as possible into FHM. As a basic example, switching among two different sets of equations as a Boolean signal changes value might be expressed as follows:

```
switch b
when False
    equations₁
when True
    equations₂
```

If the type system approach outlined in this paper is to work for FHM, we need to consider how to handle such constructs from a type perspective. This is done in Section 3.4. There are many other outstanding problems related to implementation of structurally dynamic systems. But those are outside the scope if this paper.

3 Structural types for signal relations

We now define the notion of structural type and show how it enables structural analysis to be carried out in a modular way, without having to first expand out signal relations to "flat" systems of equations. The key difficulty is abstraction of structural types, and consequently the section mostly focuses on that aspect.

3.1 The structural type

In essence, a signal relation is an *encapsulated* system of equations. When a signal function is applied, these equations impose constraints on signals in scope at the point of application through the variables of the signal relation *interface*. A larger system of equations is thus formed, composed from equations contributed by each applied signal relation.

Let us consider a simple example:

$$\begin{aligned}&\mathit{foo} :: \mathit{SR}\,(\mathit{Real}, \mathit{Real}, \mathit{Real})\\ &\mathit{foo} = \textbf{sigrel}\ (x_1, x_2, x_3)\ \textbf{where}\\ &\quad f_1\, x_1\, x_2\, x_3 = 0\\ &\quad f_2\, x_2\, x_3 \quad\;\; = 0\end{aligned}$$

Let us assume a context with five variables, u, v, w, x, y, and let us apply *foo* twice in that context:

$$\begin{aligned}&\mathit{foo} \lozenge (u, v, w)\\ &\mathit{foo} \lozenge (w, u + x\ , v + y)\end{aligned}$$

The result, obtained by substituting the variables u, v, w, x, y into the equations of *foo*, is the following system of equations:

$$\begin{aligned}&f_1\, u\, v\, w &&= 0\\ &f_2\, v\, w &&= 0\\ &f_1\, w\, (u + x\ )\, (v + y) &&= 0\\ &f_2\, (u + x\ )\, (v + y) &&= 0\end{aligned}$$

Note that each application of *foo* contributed two equations to the composed system, each for a subset of the variables to the right of the relation application operator ◊.

As discussed in Section 1, the aim is now to analyse the structure (which variables occur in which equations) of the composed system in order to identify situations that definitely will result in over- or under-determined systems of equations.

However, for a variety of reasons, it is not desirable to assume that this can be done by simply unfolding the applied relations as was done above. In the context of FHM, what goes to the left of ◊ is a signal relation *expression* that may involve parameters that are not known at compile time, thus preventing the expression from being evaluated statically. Or the exact contribution of the applied signal relation might not be known for other reasons, for example due to separate compilation or because it is structurally dynamic.

Thus, we are only going to assume that the *type* of the applied signal relation is known. To enable structural analysis, the type of signal relations is enriched by a component reflecting its structure.We refer to this as the *structural type* of the signal relation.

Definition 2 (Structural type of system of equations) The structural type of a system of equations is the incidence matrix of that system. It has one row for each equation, and one column for each variable in scope—only "unknown" signal variables are of interest here, not parameters or "known" (input) signal variables. An occurrence of a variable in an equation is indicated by 1, a non-occurrence by 0.

Note that Definition 2 concerns systems of equations. For a *signal relation*, i.e. an *encapsulated* system of equations, the structural type is limited to the equations *contributed* by the signal relation and the variables of its interface. If the interface includes records of signal variables, like *Pin* of the simple circuit example in Section 2.2, then each field counts as an independent variable. We defer a precise definition until section 3.3.

As an example, consider the signal relation *foo* above. Its type, including the structural part, is:

$$\mathit{foo} :: \mathit{SR}(\mathit{Real}, \mathit{Real}, \mathit{Real})\begin{pmatrix}1 & 1 & 1\\ 0 & 1 & 1\end{pmatrix}$$

3.2 Composition of structural types

Now let us consider composition of structural types. The overall structural type for a sequence of equations is obtained by simply joining the incidence matrices for the individual equations as the same set of variables is in scope across all equations.

The structural type for a basic equation of the form

$$e_1 = e_2$$

is a single-row matrix indicating which variables occurs in expressions e_1 and e_2.

The structural type for the second form of equation, signal relation application, is more interesting. The general form of this kind of equation is:

$sr \lozenge (e_1, e_2, \dots, e_i)$

where e_1, e_2, …, e_i are expressions over the signal variables that are in scope. These expressions and their relation to the variables in scope can also be represented by an incidence matrix, with one row for each expression and one column for each variable. The incidence matrix of the signal relation *application* is then obtained by Boolean matrix multiplication—which is understood as Boolean conjunction, $\wedge$ (logical "and"), and addition as Boolean disjunction, $\vee$ (logical "or")—of the structural type of the applied signal relation and the incidence matrix of the right-hand side expressions.

Returning to the example from the previous section, the incidence matrix of the right-hand side of the application

$foo \lozenge (u, v, w)$

in a context with five signal variables u, v, w, x, y is

$$\begin{array}{ccccc} u & v & w & x & y \\ \hline 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{array}$$

(where the columns have been labelled for clarity). Multiplying the structural type of foo with this matrix yields:

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} \begin{array}{ccccc} u & v & w & x & y \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{array} = \begin{array}{ccccc} u & v & w & x & y \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{array}$$

Similarly, for $foo \lozenge (w, u + x, v + y)$, we obtain:

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} \begin{array}{ccccc} u & v & w & x & y \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{array} = \begin{array}{ccccc} u & v & w & x & y \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{array}$$

The complete incidence matrix for the two applications of *foo* is thus

$$\begin{array}{ccccc} u & v & w & x & y \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{array}$$

Compare with the fully expanded system of equations in the previous section.

3.3 Abstraction over structural types

In the previous section, we saw how to obtain the overall structural type of a composition of signal relations given the structural types of the involved signal relations. The next step is to consider how to encapsulate a system of equations in a signal relation. It is often the case that the set of variables in the interface of a signal relation, the *interface variables* is a proper subset of the variables that are in scope. A signal relation may thus abstract over a number of *local variables*. This, in turn, means that a number of the equations at hand *must* be used to solve for the local variables: the local variables are not going to be in scope outside the signal relation, and thus it is not possible to add further equations for them later.

The available equations are thus going to be partitioned into *local equations*, those that are used to solve for local variables, and *interface equations*, those that are contributed to the "outside" when the signal relation is applied. This immediately presents an opportunity to detect instances of over- and under-determined systems of equations for the local variables on a per signal relation basis. However, it also presents a very hard problem as the partitioning is not uniquely determined, which in general implies that a signal relation does not have unique best structural type.

To illustrate, consider encapsulating the example from the previous section in a signal relation where only the variables u and y appear in the interface:

bar = **sigrel** (u, y) **where**
  $foo \lozenge (u, v, w)$
  $foo \lozenge (w, u + x, v + y)$

Recall the incidence matrix of the encapsulated system:

$$\begin{array}{ccccc} u & v & w & x & y \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{array}$$

Three of the underlying equations are needed to solve for the local variables v, w, and x, the remaining one is the interface equation. But the only equation that *cannot* be chosen as the interface equation is number 2, as no interface variable occurs in this equation. Projecting out the columns for the interface variables for the the incidence matrices for the three possible choices of interface equation yields

$$\begin{array}{cc} u & y \\ (1 & 0) \end{array} \quad \begin{array}{cc} u & y \\ (1 & 1) \end{array} \quad \begin{array}{cc} u & y \\ (1 & 1) \end{array}$$

The last two possibilities are equivalent, so this leaves us with two possible structural types: the signal relation *bar* can either provide a single equation in which the first variable of the interface occurs, or it can provide an equation in which both interface variables occurs, depending on the chosen equation partitioning in *bar*.

A modelling language compiler will decide on a specific partitioning. But this choice is typically dictated by intricate numerical considerations and often also by the usage context. As it is essential that type checking is compositional, it is clear that the partitioning must be done independently of usage context. And to ensure that the type system is independent of arbitrary implementation choices, as well as reasonably easy to understand for the end user, it is clear that the partitioning should not depend on low-level numerical considerations either.

There are two approaches for dealing with the situation. One is to *accept* that a signal relation can have more than one structural type. This paper does not explore that avenue as there is a risk that it would lead to a combinatorial explosion of possibilities to consider. Still, it should not be ruled out. The other approach is to decide on a suitable notion of "best" structural type. Then, if a signal relation has more than one possible structural type, choose the best one, if this is a uniquely determined choice, otherwise approximate all best types with a type that is better than them all, but still as informative as possible, and take this approximation as the structural type of the signal relation.

We are going to adopt a notion of "best" that reflects the observation that an equation is more useful the more variables that occur in it (as this gives more flexibility when choosing which equation to use to solve for which variable). We are further going to assume that an implementation is free to make such a best choice. The latter might not be the case, but we should then keep in mind that the objective of the type system is *not* to guarantee that a system of equations *can* be solved, but to detect cases where a system of equations definitely *cannot* be solved. Assuming a freedom of choice is thus a safe approximation.

Definition 3 (Subsumed variable)
Let V_1 and V_2 be sets of variables. V_1 is *subsumed* by V_2 iff $V_1 \setminus V_2 = \varnothing$.

Definition 4 (Subsumed structural type)
Let s_1 and s_2 be structural types. s_1 is *subsumed* by s_2 iff there exists a permutation of the rows of the incidence matrix for s_2 such that the variables of each row of the incidence matrix for s_1 are subsumed by the variables of the corresponding row of the permuted incidence matrix for s_2. The subsumed relation on structural types is denoted by the infix symbol $\leq$.

Definition 5 (Best structural types)
Let S be a set of structural types. The *best structural types* in S is the set

$$\{s \mid s \in S \wedge \neg(\exists s' \in S : s \leq s)\}$$

Returning to the signal relation *bar* above, we find that it actually has a single best structural type since

$$\begin{array}{cc} u & y \\ (1 & 0) \end{array} \leq \begin{array}{cc} u & y \\ (1 & 1) \end{array}$$

The complete type of *bar* is thus:

$$bar :: SR(Real, Real)\begin{pmatrix}1 & 1\end{pmatrix}$$

As an example of a case where there is not any best type, consider

$$s_1 = \begin{pmatrix}1 & 0 & 1 \\ 1 & 0 & 0\end{pmatrix}, \quad s_2 = \begin{pmatrix}0 & 0 & 1 \\ 1 & 1 & 1\end{pmatrix}$$

Note that $s_1 \not\leq s_2$ and $s_2 \not\leq s_1$. Neither is better than the other, and the best structural types of $S = \{s_1, s_2\}$ is S.

What is needed if there is more than one best type is to find an approximation in the form of an upper bound that subsumes them all. Clearly such a bound exists: just take the incidence matrix with all 1s, for example. That corresponds to an assumption that each equation can be used to solve for any variable, meaning that we are back to the approach of counting equations and variables. However, to avoid loosing precision unnecessarily, a *smallest upper bound* should be chosen. As the following example shows, there may be more than one such bound, in which case one is chosen arbitrarily.

Consider the two structural types:

$$\begin{pmatrix}1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1\end{pmatrix}, \quad \begin{pmatrix}0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1\end{pmatrix}$$

Upper bounds can be constructed by taking the union of the first incidence matrix and all possible row permutations of the second one. As there are only two rows, we get two upper bounds:

$$\begin{pmatrix}1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1\end{pmatrix}, \quad \begin{pmatrix}1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1\end{pmatrix}$$

Neither is smaller than the other. However, they are both as small as possible, as removing a single 1 from any matrix means it will not subsume one or the other of the original matrices. Thus, in general, the least upper bound of structural types under the subsumed ordering is not uniquely determined.

We can now give a definition of the structural type of a signal relation:

Definition 6 (Structural type of a signal relation)
The *structural type of a signal relation* with a body of m equations over n variables, of which i variables occur in the interface, if that type exists, is an $(m-(n-i))\times i$ incidence matrix that is a least upper bound of the structural types of all possible choices of interface equations.

The following algorithm determines the structural type of a signal relation when one exists, or reports an error otherwise. We claim this without proof, leaving that as future work:

Arguments

1. Structural type s for the system of equations of the body of the signal relation in the form of an $m\times n$ incidence matrix (m equations, n variables).
2. The set V of variables, $|V| = n$, and a mapping from variables to the corresponding column number of the incidence matrix.
3. The set I of interface variables of the signal relation.

Result

- If successful, an $(m-(n-|I|))\times|I|$ incidence matrix representing the structural type of the signal relation.
- Otherwise, an indication of the problem(s): under- or overdetermined system of local equations; overdetermined system of interface equations.

Algorithm

1. Let $L = V\setminus I$ be the set of local variables. Partition s into three parts:
   - s_L: rows corresponding to equations over variables in L only, the *a priori local equations*;
   - s_I: rows corresponding to equations over variables in I only the a priori interface equations;
   - s_M: remaining rows, corresponding to equations over mixed interface and local variables.

   Let m_L, m_I, m_M be the number of rows of s_L, s_I, and s_M respectively. (Note that the a priori local equations can *only* be used to solve for local variables, whereas the a priori interface equations can *only* be used to solve for interface variables.)
2. Let $k = |L| - m_L$. k is the number of equations in addition to local ones that are needed to solve for all local variables.
   a. If $k < 0$, report "overdetermined local system of equations".
   b. If $k > m_M$, report "underdetermined local system of equations".
3. Initialise S_I to $\varnothing$
4. Choose k rows from s_M in all possible ways $\binom{m_M}{k}$ possibilities, $m_M \geq k$). For each such choice:
   a. Partition s_M into $s_{L'}$ containing the k chosen rows and $s_{I'}$ containing the remaining rows.
   b. Consider s_L and $s_{L'}$ restricted to the local variables L as a bipartite graph and compute a maximum matching using the standard augmenting path algorithm [1, pp. 246–250]. Check if the size of the matching is equal to $|L|$. If yes, this means that each variable in L can be paired with a row from s_L or $s_{L'}$ in which it occurs, which is a necessary condition for using the equations corresponding to the rows from s_L or $s_{L'}$ to solve for the local variables.
   c. Consider s_I and $s_{I'}$ restricted to the interface variables I as a bipartite graph and compute a maximum matching using the standard augmenting path algorithm. Check if the size of the matching is equal to the number of rows of s_I and $s_{I'}$, i.e. $m_I + m_M - k$. If yes, then this means that all equations corresponding to the rows of s_I and $s_{I'}$ can be used simultaneously to solve for one of the interface variables. This is a necessary condition for ensuring that the interface equations contributed by the signal relation does not constitute an overdetermined system.
   d. If both checks above passed, then this particular choice of k rows is *valid*.
   e. For each valid choice, add $s_{I'}$ restricted to the variables I to $S_{I'}$.
5. If $S_{I'} = \varnothing$, it is not possible to solve for the local variables and/or the interface equations contributed by the signal relation are going to be overdetermined. Report the problem.

6. Determine the best structural types $S_{I''}$ of $S_{I'}$.
7. Let $s_{I''}$ be a least upper bound of $S_{I''}$.
8. The incidence matrix obtained by joining s_I and $s_{I''}$ is the structural type of the signal relation, i.e. a least upper bound of the structural types of all possible choices of interface equations.

3.4 Structurally dynamic systems

To conclude the development, we briefly consider how to handle structurally dynamic systems, for example of the type illustrated in section 3.3. Clearly, the structural types of the equations in the different branches could be different. However, at any point in time, the choice of which equations that are active is determined by the condition of the **switch**-construct. Thus, the structural type of the entire **switch**-construct is the *greatest lower bound* of the structural types of the branches, as that is the only thing which is guaranteed at all points in time. One may also want to impose additional consistency constraints between the branches to avoid unpleasant surprises at run-time, e.g. due to the system of equations all of a sudden becoming overdetermined. But this has not yet been investigated.

3.5 Implementation

The algorithm for computing the structural type for a signal relation has been prototyped in Haskell. It implements all aspects of the described algorithm, except that it has not been verified whether the computation of upper bounds indeed yields one of the least upper bounds. The time complexity of the algorithm is a concern. For example, the $\binom{m_M}{k}$ possible partitionings of the mixed equations that need to be investigated could, in adverse circumstances, be a large number. However, there may be ways to exploit more of the structure of the equations in order to limit the number of alternatives to consider. It is also easy to check how many partitioning there are before starting to enumerate them, and if they are judged to be too many, one can simply default to a safe over approximation of the type.

4 Structural types for a simple electrical circuit

As an example, let us apply the structural type system developed in section 3 to the simple electrical circuit from section 2.2.

Let us first consider the resistor. Recall that *Pin* is a record of two fields *v* and *i*, and that the signal relation interface thus consists of *four* variables: *p.v*, *p.i* , *n.v*, and *n.i*:

resistor :: *Resistance* → *SR* (*Pin*, *Pin*)
resistor r = **sigrel** (*p*, *n*) **where**
  twoPin ◊ (*p*, *n*, *u*)
  r * *p.i* = *u*

Before approximation, the two possible structural types for *resistor* are

$$\begin{pmatrix}0&1&0&1\\0&1&0&0\end{pmatrix},\ \begin{pmatrix}0&1&0&1\\1&0&1&0\end{pmatrix}$$

reflecting a choice between using $u = p.v - n.v$ or $r * p.i = u$ for solving for the local variable *u*. (The equation $u = p.v - n.v$ is contributed by *twoPin*. However, note that only its *structural type* is of interest here, not the exact equation.) This gets approximated with a least upper bound to:

$$\begin{pmatrix}0&1&0&1\\1&1&1&0\end{pmatrix}$$

Of course, *resistor* cannot provide a single equation in which all of *p.v*, *p.i* , and *n.v* occur. But as the equation can only be used to solve for one of the variables, and as an equation can be provided for either two of the variables or the third, this is not too bad. Let us now consider *inductor* :

inductor :: *Inductance* → *SR* (*Pin*, *Pin*)
inductor l = **sigrel** (*p*, *n*) **where**
  twoPin ◊ (*p*, *n*, *u*)
  l * **der** *p.i* = *u*

The possible structural types before approximation are the same as for resistor, but this time reflecting a choice between using $u = p.v - n.v$ or $p.i = \int p.i' dt$, where *p.i* is the state derivative, for solving for the local variables. Note that the equation $l * p.i' = u$ is local, as neither the state derivative nor *u* occurs in the interface of *inductor*. After approximation, the structural type of *inductor* becomes the same as that of *resistor* . The case for *capacitor* is also very similar, and both the possible structural types prior to approximation and the final structural type are again the same. For a final example, suppose a mistake has been made in the definition of *simpleCircuit*: instead of

connect *r1n cp*
connect *r2n lp*

the equations read

connect *r2n lp*
connect *r2n lp*

Note that the number of equations and variables remain exactly the same in the two cases (each **connect** above is expanded to one equality constraint and one sum-to-zero equation). The structural type checking algorithm presented in this paper correctly reports that *simpleCircuit* is a locally underdetermined system. If only variables and equations had been counted, this error would not have been detected.

5 Future work

It should be emphasised that what has been presented in the present paper is only a preliminary investigation into the basics of a type-based structural analysis for modular systems of equations. It is not yet yet a full-featured type system. In particular, we have only considered the structural aspect in isolation, and to that end it was tacitly assumed that the structural types of composed signal relations were known, enabling the overall structural type of signal relations to be computed in a bottom-up manner.

However, FHM aims at treating signal relations as first class entities. One consequence of this is that signal relations can be *parametrised*, including on other signal relations. In FHM, a parametrised signal relation is simply a function that computes a signal relation given values of the parameters, which could include other signal relations. The question then is how to determine the structural type of any signal relation parameters.

One option would be to insist that the structural types of signal relation parameters is always declared. This could be cumbersome, but there is always the possibility of making a permissible (imprecise) default assumption in the absence of explicit declarations. Another option might be to try to infer suitable structural constraints for the parameters from how they are being used in Hindley-Milner fashion. A third option would be to move to a framework of *dependent types* [17, 16] where types are indexed by (can depend on) *terms*. In our case, the incidence matrices that represent the structural type would be considered term-level data, and the output structural type of a parametrised signal relation is then allowed to depend on the input structural type(s), or even the values of other parameters, meaning that the output structural type will be given as a function of the parameter values.

Incidentally, Modelica effectively also provides parametrised signal relations through its mechanism of replaceable components. Here the problem is addressed by syntactically requiring a default value for the replaceable component, which is used for type checking, and additionally insisting that any replacement conforms with the type of the default value in such a way that the result after any replacement is still guaranteed to be well-typed.

Another aspect that was not considered is how to handle equations on arrays. If the sizes of the arrays are manifestly known, it would be possible to consider an array equation simply as a shorthand notation for equations between the individual elements. But that is not very attractive, and it would inevitably lead to unwieldy structural types, bloated with lots of repetitive information. And, of course, if the array sizes are not manifest but parameters of the relation, it would be even more problematic. The most feasible approach is likely to restrict array equations in such a way that each such equation can be considered a single equation for the purpose of the structural types. Again, moving to a setting of dependent types might be helpful, as the type checking depends on term-level data, i.e. the sizes of the arrays. Dependent type systems supporting explicitly sized data has been studied extensively. One good example is Dependent ML [19, 20].

We would also like to integrate checking of physical dimensions [10] into the FHM type system. We observe that this is another reason to look closer at dependent types since the types become dependent on term-level data. For example, if an entity with a dimension type is subject to iterated multiplication, the resulting dimension depends on *how many times* the multiplication was iterated.

Finally, there are usability aspects that needs to be considered. While the type errors that are reported should be attributed fairly precisely to the component that is faulty, it is not clear how to phrase the error messages such that the problem becomes evident to the end user. Also, we need to keep in mind the conservative nature of the type system: there is no guarantee that further errors will not be discovered when a complete system of equations has been assembled. Combining the approach developed here with that of Bunus and Fritzson [3, 4] might help on both counts.

6 Conclusions

This paper presented a preliminary investigation into type system for modular systems of equations. The setting of the paper is equation-based, non-causal modelling, but the central ideas should have more general applicability. The paper showed how attributing a *structural type* to equation system fragments allows over- and underdetermined system fragments to be identified separately, without first having to assemble all fragments into a complete system of equations. The central difficulty was handling abstraction of systems of equations. The paper presented an algorithm for determining the best possible type for an abstracted system, although this may involve approximation.

It should be emphasised that was has been presented is not yet a complete type system. The paper only considers the structural aspect, and it was tacitly assumed that these structural types essentially could be determined in a straightforward bottom-up manner. The goal of treating signal relations as first class entities raises a number of further challenges, some of which were discussed in Section 5.

Acknowledgements

This work was supported by EPSRC grant EP/D064554/1. The author wishes to thank the anonymous referees for many useful suggestions that helped improve the paper.

References

[1] A.V. Aho, J.E. Hopcroft, J.D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, 1983.

[2] D. Broman, K. Nyström, P. Fritzson. *Determining over- and under-constrained systems of equations using structural constraint delta*. In GPCE '06: Proc. 5th Int. Conference on Generative Programming and Component Engineering, pp. 151–160, Portland, Oregon, USA, 2006. ACM.

[3] P. Bunus, P. Fritzson. *A debugging scheme for declarative equation based modeling languages*. In Proc. 4th Int. Symposium on Practical Aspects of Declarative Languages (PADL 2002), vol. 2257 of Lecture Notes in Computer Science, pages 280– 298, Portland, OR, USA, January 2002. Springer-Verlag.

[4] P. Bunus, P. Fritzson. *Methods for structural analysis and debugging of Modelica models*. In Proc. 2nd Int. Modelica Conference, pp. 157– 165, Oberpfaffenhofen, Germany, March 2002.

[5] A. Courtney, H. Nilsson, J. Peterson. *The Yampa arcade*. In Proc. 2003 ACM SIGPLAN Haskell Workshop (Haskell'03), pp. 7–18, Uppsala, Sweden, August 2003. ACM Press.

[6] A. L. Dulmage, N. S. Mendelsohn. *Coverings of bipartite graphs*. Canadian Journal of Mathematics, 10:517–534, 1958.

[7] H. Elmqvist. *A Structured Model Language for Large Continuous Systems*. PhD thesis TFRT-1015, Department of Automatic Control, Lund Institute of Technology, 1978.

[8] G. Giorgidze, H. Nilsson. *Switched-on Yampa: Declarative programming of modular synthesizers*. In P. Hudak, D.S. Warren, eds., Practical Aspects of Declarative Languages (PADL) 2008, vol. 4902 of Lecture Notes in Computer Science, pp. 282–298, San Francisco, CA, USA, January 2008. Springer-Verlag.

[9] P. Hudak, A. Courtney, H. Nilsson, J. Peterson. *Arrows, robots, and functional reactive programming*. In J. Jeuring, S.P. Jones, eds., Advanced Functional Programming, 4th International School 2002, vol. 2638 of Lecture Notes in Computer Science, pages 159–187. Springer-Verlag, 2003.

[10] A. Kennedy. *Dimension types*. In Proc. 5th European Symposium on Programming, no. 788 in Lecture Notes in Computer Science. Springer-Verlag, 1994.

[11] Modelica Association. *Modelica – A Unified Object--Oriented Language for Physical Systems Modeling: Tutorial* version 1.4, December 2000.

[12] Modelica Association. *Modelica – A Unified Object--Oriented Language for Physical Systems Modeling: Language Specification Version 3.0*, September 2007.

[13] H. Nilsson, A. Courtney, J. Peterson. *Functional reactive programming, continued*. In Proc. 2002 ACM SIGPLAN Haskell Workshop (Haskell'02), pp.51–64, Pittsburgh, Pennsylvania, USA, Oct. 02. ACM Press.

[14] H. Nilsson, J. Peterson, P. Hudak. *Functional hybrid modeling*. In Proc. PADL'03: 5th Int. Workshop on Practical Aspects of Declarative Languages, vol. 2562 of Lecture Notes in Computer Science, pages 376–390, New Orleans, Louisiana, USA, January 2003. Springer-Verlag.

[15] H. Nilsson, J. Peterson, P. Hudak. *Functional hybrid modeling from an object-oriented perspective*. In P. Fritzson, F. Cellier, and C. Nytsch-Geusen, eds., Proc. 1st Int. Workshop on Equation-Based Object-Oriented Languages and Tools, no. 24 in Linköping Electronic Conference Proceedings, pages 71–87. Linköping University Electronic Press, 2007.

[16] B.C. Pierce. *Types and Programming Languages*. MIT Press, 2002.

[17] S. Thompson. *Type Theory and Functional ProgramProgramming*. Addison-Wesley, 1991.

[18] Zhanyong W., P. Hudak. *Functional reactive programming from first principles*. In Proc. PLDI'01: Symposium on Programming Language Design and Implementation, pp. 242–252, June 2000.

[19] Hongwei Xi, F. Pfenning. *Eliminating array bound checking through dependent types*. In Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 249–257, Montreal, June 1998.

[20] Hongwei Xi, F. Pfenning. *Dependent types in practical programming*. In Proc. ACM SIGPLAN Symposium on Principles of Programming Languages, pages 214–227, San Antonio, January 1999.

Corresponding author: Henrik Nilsson,
School of Computer Science
University of Nottingham, United Kingdom,
nhn@cs.nott.ac.uk

Accepted EOOLT 2008, June 2008
Received: September 5, 2008
Accepted: November 29, 2008

Beyond Simulation: Computer Aided Control System Design using Equation-based Object-oriented Modelling for the Next Decade

Francesco Casella, Filippo Donida, Marco Lovera, Politecnico di Milano, Italy

{casella, donida, lovera}@elet.polimi.it

SNE Simulation Notes Europe SNE 19(1), 2009, 29-41, doi: 10.11128/sne.19.tn.09925

After 20 years since their birth, equation-oriented and object-oriented modelling techniques and tools are now mature, as far as solving simulation problems is concerned. Conversely, there is still much to be done in order to provide more direct support for the design of advanced, modelbased control systems, starting from object-oriented plant models. Following a brief review of the current state of the art in this field, the paper presents some proposals for future developments: open model exchange formats, automatic model-order reduction techniques, automatic derivation of simplified transfer functions, automatic derivation of LFT models, automatic generation of inverse models for robotic systems, and support for nonlinear model predictive control.

Introduction

Control system engineering requires to master the dynamics of plants which are in general complex, interacting, multi-physics and multi-disciplinary. This explains why object-oriented modelling (OOM) and a-causal, equationbased, object-oriented languages (EOOL) always had a very strong connection with control system design. It is by no means accidental that much pioneering work in the OOM field was carried out within systems and control departments and research groups: consider, for example, the Omola language and the associated OmSim simulation environment, developed at the Department of Automatic Control of Lund Technical University [29, 30, 4], or the MOSES environment developed at the Dipartimento di Elettronica of Politecnico di Milano [26, 9]. During the '90, OOM was considered a very promising tool for Computer Aided Control System Design (CACSD), and there was a lot of activity in this field, which eventually culminated in the development of the Modelica Language [32].

At the beginning of that decade, papers appeared on the subject in the IEEE's Control Systems Magazine [31, 10], which discussed the potential of OOM for control system design. Reading those papers in retrospect shows that some of the promises where actually met or even exceeded: OOM is now a mature field, both from a theoretical side and from the point of view of available simulation tools. On the other hand, much work still has to be done on two fronts. The first one, which has a more "political" nature, is spreading the OOM culture among in the control engineering community, which is still largely dominated by block-oriented modelling, and by the (mis)use of MatLab/Simulink for physical systems modelling; this challenge is of paramount importance, but it out of the scope of this paper. The second one, instead, is to develop tools which allow to use EOOL models and tools not only for simulation, but also for the design of advanced control systems. The availability of such tools is crucial in order to narrow the gap between the large body of highly sophisticated control theory developed during the last 20 years, and the application of this theory to real-life cases, beyond textbook-sized examples. This is the topic of the present paper.

Given the background and the past experience of the authors, the discussion might be biased towards the Modelica language and related tools. However, strictly object-oriented features such as inheritance, encapsulation and hierarchical composition do not play any significant role in the analysis and proposals made within this paper, which essentially focuses on transformations of flattened models. On the contrary, the discussion is relevant for any equationbased modelling language, provided that it is a-causal and it allows symbolic manipulation of the equations by the compiler.

The paper is structured as follows: Section 1 gives a high-level view of the modelling activities required for control system design, while the following Section 2 discusses how currently available tools can help the control engineer in his/her task, with particular reference to Modelica tools. Sections 3 and 4, which are the core of the paper, propose several research and development directions to substantially increase the level of support to the control engineer, willing to apply advanced control theory to real-life problems. Section 5 concludes the paper with final remarks.

1 The role of mathematical models in control system design

The design of control systems always requires some knowledge about the dynamic behaviour of the plant under control. When the plant design is mature and well-known, and the control system design is based on Proportional-Integral-Derivative (PID) controllers, the latter is often based on past experience and possibly on some empirical measurements. In this case, which covers the vast majority of installed industrial controllers, no (explicit) dynamical modelling is needed.

On the other hand, in an increasing number of cases, the performance of the control system is becoming a key competitive factor for the success of innovative, high-tech systems. To name a few examples, consider high-performance mechatronic systems (such as robots), vehicles enhanced by active integrated stability, suspension, and braking control, aerospace system, advanced energy conversion systems. All these cases possess at least one of the following features, which call for some kind of mathematical modelling for the design of the control system:

- closed-loop performance critically depends on the dynamic behaviour, which is not well-known in advance;
- the system is complex, made of many closely interacting subsystems, so that the behaviour of the whole system is more than just the sum of its parts;
- advanced control systems are required to obtain competitive performance, and these in turn depend on explicit mathematical models for their design;
- the system is very expensive and/or safety critical, requiring extensive validation of good control performance by simulation.

In most of these cases, two different classes of mathematical models are derived: compact models for control design and detailed models for system simulation.

1.1 Compact models for control design

Models belonging to this class are directly used for controller design, and are usually formulated in state-space form:

$$\begin{aligned}\dot{x}(t) &= f(x(t),u(t),p,t)\\ \dot{y}(t) &= g(x(t),u(t),p,t)\end{aligned} \tag{1}$$

where x is the vector of state variables, u is the vector of system inputs (control variables and disturbances), y is the vector of system outputs, p is the vector of parameters, and t is the continuous time. A special case is that of linear, time-invariant models (LTI), which can be described as:

$$\begin{aligned}\dot{x}(t) &= Ax(t)+Bu(t)\\ \dot{y}(t) &= Cx(t)+Du(t)\end{aligned} \tag{2}$$

or, equivalently, as a transfer function:

$$G(s) = C(sI-A)^{-1}B+D \tag{3}$$

In many cases, the dynamics of systems in the form (1) is approximated by (2) via linearization around some equilibrium point. There is also a vast body of advanced control techniques which are based on discrete-time models:

$$\begin{aligned}x(k+1) &= f(x(k),u(k),p,k)\\ y(k) &= g(x(k),u(k),p,k)\end{aligned} \tag{4}$$

where the integer time step k usually corresponds to the sampling time T_s of a digital control system. Many techniques are available to transform (1) into (4).

These models must capture the fundamental dynamics which is relevant for control system performance, while remaining as simple as possible: most advanced control design techniques start to become intractable for systems of order greater than about ten. If the models are simple enough, it is also sometimes possible to express the dependence of key dynamic features (such as, e.g., the natural frequency and damping coefficient of an oscillating dynamics) from plant design data. This can be very important to assess the impact of physical system design decisions on controller performance. For example, if the natural frequency of the first mode of oscillation limits the controller bandwidth, and it is found that this frequency mainly depends on the stiffness of a certain mechanical component, then it might be reasonable to change the mechanical design of that component in order to improve the overall performance.

In order to derive such simple models, it is usually necessary to introduce many, sometimes drastic, simplifying assumptions: all those phenomena that only marginally affect the equilibrium values and/or the control-relevant dynamics of the system are neglected. This activity requires highly skilled and experienced modellers, with a good knowledge of control design techniques, as well as of domain-specific strategies for model simplification.

1.2 Detailed models for system simulation

At the other end of the modelling spectrum, detailed simulation models can be found. Although it is always necessary to make reasonable modelling assumptions (a model is always a focused and limited description of the physical world), simulation models can include a lot more detail and second-order effects, since modern CPUs and simulation environments can easily handle complex systems with (tens of) thousands of variables. It is well-known that OOM methodologies and EOOLs provide very good support for the development of such models, thanks to equation-based modelling, a-causal physical ports, aggregation and inheritance. If the OOM model does not contain discrete variables and events, then it is basically equivalent to the set of DAEs:

$$F(x(t),\dot{x}(t),u(t),y(t),p,t)=0 \qquad (5)$$

Many EOOLs and tools also allow to describe hybrid systems, with discrete variables, conditional equations or expressions, and events. For example, see [7, 8] and references therein for hybrid system descriptions based on hybrid automata, or the Modelica language specification [41], in particular Appendix C. Although hybrid system control is an interesting and emerging field, for the sake of conciseness this paper will focus on purely continuous-time physical models, with application to the design of continuous-time or sampled-time control systems.

These larger, more detailed models play a double role, with respect to those described in the previous sub-section. On one hand they allow to check how good (or crude) the compact models is, compared to a more detailed description, thus helping to develop good compact models. On the other hand, they allow to check the closed-loop performance of the controlled system, once a controller design is available. It is in fact well-known that validating the closed-loop performance using the same simplified model that was used for control system design is not a sound practice; conversely, validation performed with a more detailed model is usually deemed a good indicator of the control system performance, whenever experimental validation is not possible for some reason.

2 Overview of current CACSD practice with EOOLs

As of today, the practising control engineer already gets much support from EOOL-based tools for his/her control system design activities.

2.1 Support to control system synthesis

A typical starting point for the design of the control system is the analysis of the linearized dynamics of the plant, around one (or more) steady-state operating conditions. If the EOOL tool only supports simulation, then one can run open-loop simulations of the plant model, subject to step or to, e.g., pseudo-random binary sequence inputs, and then reconstruct the dynamics by system identification procedures.

A more direct approach, supported by many tools, is to directly compute the A,B,C,D matrices of the linearized system around specified equilibrium points, using symbolic and/or numerical techniques. The result is usually a high-order linear system, which can then be reduced to a low-order system by standard techniques for linear model order reduction, such as, e.g., balanced truncation.

A non-trivial issue with both approaches is the computation of the equilibrium point (what is sometimes called DC analysis in the field of electrical circuit simulation). In a typical setting, the desired steady-state values of the outputs $\bar{y}$ are known, and the tool must solve the steady-state initialization problem for the system (5):

$$F(\bar{x},0,\bar{u},\bar{y},p,0)=0 \qquad (6)$$

in order to find out the corresponding equilibrium values of the inputs $\bar{u}$ and of the states $\bar{x}$. This problem can be numerically challenging, because it often requires solving large systems of coupled nonlinear equations by iterative methods, which might fail if the iteration variables are not properly initialized. Currently available OOM tools (and, in particular, Modelica tools) are still far from providing general robust solutions to this problem. A sub-optimal approach to find equilibrium points is to initialize system (5) by giving tentative initial values to the state variables (which makes the initialization problem easier to solve) and then to simulate it until it reaches a steady state. If the system is asymptotically stable and the inputs $\bar{u}$ are known, this is relatively straightforward; otherwise, it is necessary to add suitable feedback controllers to drive the outputs to the desired values $\bar{y}$ and/or to stabilize the system. In both cases, the simulation of this initialization transient might fail for numerical reasons before reaching the steady state, due to a bad choice of the initial states.

2.2 Closed-loop performance assessment by simulation

Regardless of the actual design methodology, once the controller has been set up, an OOM tool can be used to run closed-loop simulations, including both the plant and the controller model. Many OOM tools provide model export facilities, which allow to connect an OO plant model with only causal external connectors (actuator inputs and sensor outputs) to a causal controller model in a causal simulation environment. From a mathematical point of view, this corresponds to reformulating (5) in state space form (1), by means of analytical and/or numerical transformations.

2.3 Development of simplified models

The object-oriented approach, and in particular replaceable components, allows to define and to manage families of models of the same plant with different levels of complexity, by providing more or less detailed implementations of the same abstract interfaces. For example, consider a heat exchanger model: the abstract interface has four fluid connectors, two for the hot fluid inlet and outlet, and two for the cold fluid inlet and outlet. The corresponding implementation might range from a very simple static model based on log-mean temperatures, with a few algebraic equations, up to a very detailed finite volume model using nonlinear fluid properties and empirical correlations for heat transfer, and with dozens of state variables and a few hundred algebraic equations.

This feature of OOM allows developing simulation models with different degrees of detail (and CPU load) throughout the entire life of an engineering project, from preliminary design down to commissioning and personnel training, within a coherent framework. However, this activity is based on manual work by the modeller, who needs to develop the different implementations explicitly. Moreover, it is often not easy to obtain compact models such as (1), because this requires applying simplifications that may not fit well the abstract component boundaries.

2.4 Generation of real-time simulation code

An important step in the development of embedded control systems is Hardware-In-the-Loop simulation (HIL), where the real control hardware is tested by connecting it to a realtime simulator, instead of the real plant. Many currently available EOOL-based tools support automatic generation of efficient real time code starting from fairly large simulation models in the form (5). A common strategy for this purpose is to apply inline integration [12, 11] to (5), i.e. to substitute the derivatives with their approximation formulae (e.g. Euler's formula), and then solve the system using all available numerical and symbolic techniques.

In order to provide real-time code which is fast enough, it is usually important to reduce the model complexity with respect to off-line simulation models - this can be done by following the approach sketched in Section 2.3.

2.5 Optimization

Some EOOL and tools support some kind of optimization, which might be useful for control system design. For example, the gPROMS language [6] has allowed declaring mixed-integer nonlinear optimization problems for a long time. More recently, extensions to the Modelica language were proposed to formulate optimization problems [2].

2.6 Further perspectives

It is the authors' view that EOOL-based tools should support advanced control system design problems in a much more direct way, by making extensive use of control-oriented symbolic manipulation techniques. Ideally, it would be good if the control engineer could develop a detailed simulation model by using object oriented tools and re-usable model libraries, then automatically obtain simplified, compact models which are already formulated as required by the specific control technique. The availability of such tools might promote the application of advanced, model-based techniques that are currently limited by the model development process.

Being aware that this is a very long-term goal, which might even require some kind of artificial intelligence, some first steps in this direction are discussed in the following sections, with particular reference to the Modelica language and Modelica compliant tools.

3 Basic enabling technologies

The advanced, control-oriented features of future EOOL tools need some basic enabling technologies and methodologies to build upon. These are briefly discussed in this section.

3.1 Open standards for model and data exchange

Advanced applications of OOM to control system design will most likely require using different specialized tools in a coordinated fashion, rather than relying on one-fits-all comprehensive software tools. In fact, during the last decades, the number and the quality of simulation, design and analysis tools has increased enormously: there is plenty of open and closed source software for the simulation of physical systems, control synthesis, data analysis, test, validation, personnel training via a graphical user interface, etc. Some of these tools are developed for specific purposes, while others are more general in scope (e.g., symbolic manipulation tools, differential equation solvers, data analysis packages). Unfortunately, all this software development activity did not follow any standardisation process, leading to a great diversity in the representation of the information. The definition of standard interfaces will be useful for the information exchange between different applications; as a consequence, by providing a representation for all the stages of the model manipulation (starting from the translation, going to the flattening, to the model order reduction and so forth) it will be possible to make all the applications interact at different levels, thus combining positive effects from different applications and obtaining better results.

Exchange formats for model equations and for simulation data should probably be based on the XML language, for several reasons:

- the tree structure of XML documents easily allows to represent complex data structures, including symbolic representations of equations;
- XML documents can be read with standard text-editors and browsers, thus avoiding all the problem usually raised by obscure, ad-hoc binary formats;
- there exists a large base of software (open source and commercial) for the handling of XML files;
- by re-using this existing software, it is quite straightforward to translate an XML document representing a mathematical model into any other equation-based language, and vice-versa;
- binary XML formats can be used to reduce the verbosity of XML documents and the cost of parsing them;
- there exist some languages (e.g. DTD and XSD) to formally specify the structure of the information the XML file must contain.

Such standard interfaces for flattened Modelica models and their corresponding simulation data are currently being investigated at Politecnico di Milano using the OpenModelica compiler [16, 1] as a host EOOL environment, and symbolic manipulation tools such as Mathematica, Maple or Maxima as target environments. If the model is purely continuous-time, i. e., it is equivalent to the DAE (5), then MathML [42] on one side, and ModelicaXML [35] on the other side might constitute good starting points. If hybrid models are considered, one may consider all the languages developed for the description of hybrid automata in recent years [8], even though the class of hybrid systems, which can be described in Modelica with `when` statements, is larger than just hybrid automata.

3.2 Model Order Reduction

Another key enabling technology is represented by mixed numerical-symbolic Model Order Reduction (MOR) techniques. These have already been successfully applied to the analysis of electrical circuit models, which are based on DAE models such as (5), see [40, 17], and are currently available in commercial tools such as Analog Insydes [13]. The MOR strategies are based on the clever application of three fundamental steps:

- specify an allowed error bound, e.g. in terms of percentage error of certain steady-state output values corresponding to given constant inputs, or in terms of maximum deviation of some outputs from a reference trajectory obtained with given input signals, or in terms of maximum error of small-signal frequency responses around a certain operating point and within a given frequency interval;
- derive a ranking of all terms in all equations, expressing how much each term has a significant effect on the required modelling accuracy;
- remove all terms in ascending order, until the specified error bound is reached.

Other MOR techniques exist to reduce large linear systems, based on concepts such as modal analysis and projection methods; see [38] for a comprehensive overview.

The application of such MOR tools and techniques, possibly with extended functionality and algorithms, looks very promising not only for the simplification of electrical circuit models, but also for the order reduction of generic, nonlinear DAE models, obtained from the flattening of generic EOOL models. This kind of techniques should allow to automatically obtain approximated compact models such as (1), starting from much more detailed simulation models, by formulating specific approximation bounds in control-relevant terms (e.g., percentage errors of steady-state output values, norm-bounded additive or multiplicative errors of weighted transfer functions, or l_∞-norm errors of output transients in response to specified input signals). Given the ever-increasing computation power that can be expected by Moore's law, the future of these techniques for CACSD applications definitely appears bright.

3.3 Reliaeble steady-state initialization and static model inversion

A reliable support to the control engineer's activity requires to improve the techniques to solve the steady-state equations (6), which are usually the starting point for any kind of analysis, including MOR. As pointed out earlier, solving (6) requires iterative methods which might fail if not properly initialized. Troubleshooting can be very frustrating and time-consuming, and calls for experts of both simulation methods and domain-specific models. This is not acceptable in the envisioned framework, which is based on *automatic* manipulation by EOOL tools.

One option, which is currently being investigated at Politecnico, is to introduce extensions to the Modelica language to support homotopy methods, in a way similar to the approach followed by the SPICE circuit simulation program. The basic idea is that each model has two versions: the "easy" one, for which it is easier to find a steady-state solution, and the "true" version, which is the model to be actually used for simulation. The two models share the same variables, but use different equations. The system model obtained by the aggregation of the "easy" models is represented by

$$F_e(x,\dot{x},y,u,p,t)=0 \tag{7}$$

while the aggregation of the "true" models leads to

$$F_t(x,\dot{x},y,u,p,t)=0 \tag{8}$$

The idea is now to first solve the initialization problem for (7), which should not give rise to significant numerical problems. The solution to this simplified problem constitutes the first guess for a new problem:

$$(1-\alpha)F_e(\bar{x},0,\bar{u},y,p,0)+\alpha F_t(\bar{x},0,\bar{u},y,p,0) \tag{9}$$

which will be solved by varying α from 0 to 1 in small steps, eventually finding the steady-state solution of system (8). In general, this approach should help to reduce (and hopefully eliminate) the need to manually set initial guess values for iteration variables of initialization problems.

4 New functionalities for control system design

4.1 Simplified symbolic transfer functions

In many interesting cases, the performance of the control system is limited by the dynamic behaviour of the controlled plant. For example, poorly damped oscillations can limit the bandwidth of motion control systems, as well as non-minimum phase behaviour. The control engineer can gain a lot of useful insight from approximated transfer functions, where the dependence of the critical dynamic features from a few physical parameters is clearly visible. For instance, the natural frequency of a pair of complex poles in a mechanical system might depend mainly on the stiffness and on the mass of a certain physical component, or, the time constant of a right-half-plan zero in a fluid system might depend on the fluid velocity in a certain point.

This is a first case where automatic MOR techniques could prove extremely useful. Ideally, the user should specify the steady-state operating point, the relevant inputs and outputs, and some frequency-weighted error bounds, and get low-order approximated transfer functions of the linearized system, with approximated but explicit dependence of the transfer function features (gains, poles and zeros) from the physical model parameters. A suitable combination of EOOL tools (equipped with model import/export interfaces) with existing MOR tools like Analog Insydes [13] could provide very interesting results in this direction without too much effort.

4.2 Automatic derivation of LFT models

Once a model has been reduced to a low-order state-space form by the combined application of symbolic MOR techniques and clever model simplifications as explained in Section 3.3, it might be useful to automatically bring them in the form required for advanced control system design, using symbolic manipulation tools. Modern control theory provides methods and tools in order to deal with design problems in which stability and performance have to be guaranteed also in the presence of model uncertainty, both for regulation around a specified operating point and for gain scheduled control system design.

Most of the existing control design literature assumes that the plant model is given in the form of a Linear Fractional Transformation (LFT) (see, e.g., [46, 27]), a modelling paradigm which is currently an active research topic in the control engineering and system identification communities. In the robust control framework LFT models consist of a feedback interconnection between a nominal LTI 39 plant and a (usually norm-bounded) operator which represents model uncertainties, e. g., poorly known or time-varying parameters, nonlinearities, etc. A generic such LFT interconnection is depicted in Figure 1, where the nominal plant is denoted with P and the uncertainty block is denoted with Δ. Note that this representation is extremely general, and by no means limited to uncertain LTI systems; in fact, it is possible to describe any nonlinear DAE system by putting all the nonlinear functions in the Δ block and by providing an LTI model with direct feedthrough terms to describe the algebraic equations.

LFT models can be used for the design of robust and gain scheduling controllers, but they can also serve as a basis for structured model identification techniques, where the uncertain parameters that appear in the feedback blocks are estimated based on input/output data sequences.

The process of extracting uncertain parameters from the design model of the system to be controlled is a highly complex one. Symbolic techniques play a very important role in this process: the main use for such techniques is to find, via suitable pre-processing steps, equivalent representations of rationally dependent parametric matrices, which automatically lead to lower-order LFT representations. Tools already exist to perform this task [27].

The LFT modelling problem in its simplest form is associated with the problem of designing a controller for operation near a nominal operating point for the system. The problem is then formulated on a local linearised representation of the plant to be controlled and is familiarly termed "pulling out the Δs", i.e., it consists of manually or symbolically manipulating the linearised equations in order to separate the nominal part of the plant from the uncertain one, arranging them in a suitable feedback interconnection. This reformulation of the plant model lies at the vary basis of modern robust control theory and is currently supported by a number of different symbolic manipulation tools. A recent overview of the state-of-the-art in this research area can be found in [18]. As an example, consider a timeinvariant, nonlinear state-space system in the form

$$\begin{aligned} \dot{x}(t) &= f(x(t), u(t), p) \\ y(t) &= g(x(t), u(t), p) \end{aligned} \tag{10}$$

where p denotes a vector of uncertain parameters, and assume that the equilibrium condition $\bar{x}$, $\bar{u}$, $\bar{y}$, which solves the steady-state equations

$$\begin{aligned} 0 &= f(\bar{x}, \bar{u}, p) \\ y &= g(\bar{x}, \bar{u}, p) \end{aligned} \tag{11}$$

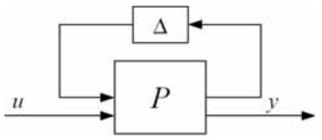


Figure 1. Block diagram of the typical LFT interconnection adopted in the robust control framework.

is available. Defining now the *deviation variables*

$$\delta x(t) = x(t) - \bar{x} \tag{12}$$

$$\delta u(t) = u(t) - \bar{u} \tag{13}$$

$$\delta y(t) = y(t) - \bar{y} \tag{14}$$

it is possible to approximate the dynamics of (10) with a the following linear, parameter-dependent system

$$\begin{aligned} \dot{\delta x}(t) &= A(p)\delta x + B(p)\delta u \\ \delta y(t) &= C(p)\delta x + D(p)\delta u \end{aligned} \tag{15}$$

where the four matrices A, B, C, D are the Jacobians of the two functions f and g:

$$\begin{aligned} A(p) &= \tfrac{\partial f}{\partial x}, \quad B(p) = \tfrac{\partial f}{\partial u} \\ C(p) &= \tfrac{\partial g}{\partial x}, \quad D(p) = \tfrac{\partial g}{\partial u} \end{aligned}$$

Under suitable assumptions (such as that the state space matrices are polynomial or rational functions of the elements of p, see, e.g., [46]) it is possible to transform the system description (15) into an LFT representation (see, again, Figure 1). As mentioned previously, converting (15) into an LFT with a Δ block of minimum dimension is a non-trivial symbolic manipulation problem.

An even more challenging formulation of the LFT modelling problem is the one of *simultaneously* representing in LFT form all the linearisations of interest for control purposes of the given nonlinear plant. Indeed, in many control engineering applications a single control system must be designed to guarantee the satisfactory closed-loop operation of a given plant in many different operating conditions in the presence of parametric and possibly non parametric uncertainty. The gain scheduling approach to the problem, which has been part of the engineering practice for decades, can be roughly summarised as follows: find one or more *scheduling variables* α which can completely parametrise the operating space of interest (e.g., the flight envelope in the case of aircraft control) for the system to be controlled; define a parametric *family* of linearised models for the plant associated with the set of operating points of interest;

finally, design a *parametric* controller which can both ensure the desired control objectives in each operating point and an acceptable behaviour during (slow) transients between one operating condition and the other. Many design techniques are now available for this problem (see, e.g., [5, 22, 37]), which can be reliably solved, provided that a suitable model in parameter-dependent form has been derived for the system to be controlled. The goal here would be to arrive at a representation of the dynamics of the nonlinear system in the form depicted in Figure 2, which is usually denoted as an LPV-LFT system, where the LPV acronym stands for Linear Parameter-Varying. The model structure now includes two feedback interconnections: the block $\Delta(p)$ takes into account the presence of the uncertain parameter vector p, while the block $\Theta(\alpha)$ models the effect of the varying operating point, parametrised by the vector of time-varying parameters α.

The state-of-the-art of modelling for gain scheduling can be briefly summarised by defining two classes of modelling approaches: *analytical* methods based on the availability of (relatively) reliable nonlinear equations for the dynamics of the plant, from which suitable control-oriented representations can be derived (see, e.g., [28] and the references therein); *experimental* methods based entirely on identification, i.e., aiming at deriving LPV models for the plant directly from input/output data (see, among many others, [21, 45, 23]). The methods belonging to the first class aim at developing LPV models for the plant to be controlled by resorting to, broadly speaking, suitable extensions of the familiar notion of linearisation, developed in order to take into account off-equilibrium operation of the system. As far as experimental methods are concerned, most LPV identification techniques are based on the assumption that the identification procedure can rely on one *global* identification experiment in which both the control input and the scheduling variables are (persistently) excited in a simultaneous way. This assumption may not be a reasonable one in many applications, in which it would be desirable to try and derive a parameter-dependent model on the basis of *local* experiments only, i.e., experiments in which the scheduling variable is held constant and only the control input is excited. Such a viewpoint has been considered in [43, 34, 23], where numerical procedures for the construction of parametric models for gain scheduling on the basis of local experiments and for the interpolation of local controllers have been proposed.

To our best knowledge the only documented attempt at deriving control-oriented LFT models automatically from a nonlinear simulator is presented in [44], where the focus was on the automatic generation of LFT models for aerospace applications. Much remains to be done. An EOOL-based CACSD tool dealing with the generation of control-oriented LFT models should allow to specify some error bounds for the system approximation (with respect to steady-state, transient, and frequency response), the choice of input, output and scheduling variables, and the choice of parameters to include in the LFT representation. Based on that, it should be able to automatically compute the structure of the interconnections defined in Figures 1 and 2 for the robust and gain-scheduling control design problems, respectively, the state-space matrices of the nominal part P of the model (either as analytical expressions, if possible, or at least as algorithms for their computation) and analytical or algorithmic representations of the feedback blocks $\Theta(\alpha)$ and $\Delta(p)$. Finally, it is apparent from the short literature review presented above that currently only physical and black-box modelling methods are available, while no general purpose CACSD tools capable of combining first principles models and experimental data in a single control-oriented model seem to exist. The convergence of the two modelling approaches both in terms of methods and tools would be a very desirable outcome of the research in this field.

4.3 Automatic computation of inverse models for robotic systems

The design of controllers for non-redundant robotic manipulators with N degrees of freedom usually starts from the equations of motion obtained from the Euler-Lagrange equations [39]:

$$B(q)\ddot{q} + H(q,\dot{q})\dot{q} + g(q) = \tau \tag{16}$$

$$y_p = K(q) \tag{17}$$

$$y_v = \frac{\partial K}{\partial q}\dot{q} \tag{18}$$

where q is the N-element vector of Lagrangian coordinates, which usually correspond to the rotation angles of the actuator motors, $\dot{q}$ is the vector of the corresponding generalized velocities, y_p describes the position and orientation vector of the end effector, y_v contains the corresponding generalized velocities, τ is the vector of generalized applied forces corresponding to each degree of freedom (usually the torques applied by rotating actuators), $B(q)$ is the inertia matrix, $H(q,\dot{q})$ is the matrix corresponding to the centripetal, Coriolis, and viscous friction forces, while the vector $g(q)$ accounts for the effects of the gravitational field; all vectors have dimension N.

The classical approach to write (16) requires to compute the so-called direct kinematics (DK), i.e. how the values of q and $\dot{q}$ translate into the position and motion of the robot's end effector, then to compute the Lagrange function, i.e. the difference between kinetic and potential energy, and apply the Euler-Lagrange equations. This can be done manually, or using one of the specialized tools available for this task. Equations (16)-(18) can then be used as a basis for both controller design and system simulation.

Within an OOM approach, it is possible to save much time by developing an object-oriented model using an EOOL, e.g. using the Modelica MultiBody library [33]. Due to the kinematic constraints imposed by the joints, the original flattened model corresponds to an index-3 DAE,

$$F(x,\dot{x},y,u)=0 \tag{19}$$

which is mathematically equivalent to the Lagrange model (16)-(18).

Currently available Modelica tools tackle the problem by applying specialized algorithms, which exploit the knowledge of the topology of the kinematic chain, as well as standard techniques such as BLT partitioning, tearing, dummy derivatives and symbolic solution of equations [33]. From a conceptual point of view, a change of state variables x allows to transform (19) into an index-1 system

$$F_1(x,\dot{x},y,u)=0 \tag{20}$$

where

$$x=\begin{bmatrix} x_p \\ x_v \end{bmatrix}=\begin{bmatrix} q \\ \dot{q} \end{bmatrix}, \quad y=\begin{bmatrix} y_p \\ y_v \end{bmatrix}, \quad u=\tau \tag{21}$$

Eventually, efficient procedures are produced to solve (20) for $\dot{x}$ and y given x and u, thus actually bringing the system into state-space form:

$$\begin{aligned} \dot{x} &= f(x,u) \\ y &= g(x,u) \end{aligned} \tag{22}$$

This formulation can be used to solve simulation problems, by linking it to any ODE/DAE solver. However, there are several other interesting things that could be done with (20), from a control engineer's perspective.

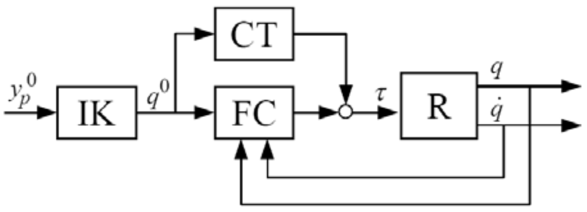


Figure 3. Block diagram of computed torque control.

Robot trajectories are originally defined in terms of end effector coordinates as functions of time $y_p^0(t)$. In order to obtain the corresponding reference trajectories in Lagrangian coordinates for the low-level robot joint controllers, (17)-(18) must be solved for q, $\dot{q}$, thus computing the so-called inverse kinematics (IK):

$$q^0 = K^{-1}(y_p^0) \tag{23}$$

$$\dot{q}^0 = \left(\frac{\partial K}{\partial q}\right)^{-1} y_v^0 \tag{24}$$

note that the Jacobian of $K(q)$ is also needed to solve (23), since analytical inverses cannot usually be obtained. Furthermore, two interesting approaches to model-based robot control are based on suitable manipulations of eq. (16): the *pre-computed torque* approach and the *inverse dynamics* approach [39].

The pre-computed torque approach is a feed-forward compensation scheme, where the theoretical torque required to follow the reference trajectory is directly fed to the torque actuators (see Figure 3) in order to obtain a good dynamic response to the set point $y_p^0(t)$. The CT block performs this task by solving (16) for τ, given the reference trajectory and its derivatives:

$$\tau = B(q^0)\ddot{q}^0 + H(q^0,\dot{q}^0)\dot{q}^0 + g(q^0) \tag{25}$$

A feedback controller (FC) is also included to deal with uncertainties and disturbances.

Since the inertia matrix B is structurally non-singular, it is always possible to solve (26) for v:

$$v = B^{-1}(q)(\tau - H(q,\dot{q})\dot{q} - g(q)) \tag{26}$$

Plugging v in the robot dynamics equation (16), one obtains:

$$\ddot{q} = v \tag{27}$$

The block diagram interpretation of these equations is shown in Figure 4: thanks to the dynamic inversion (DI) block, the dynamic relationship between the virtual input v and the Lagrangian positions and

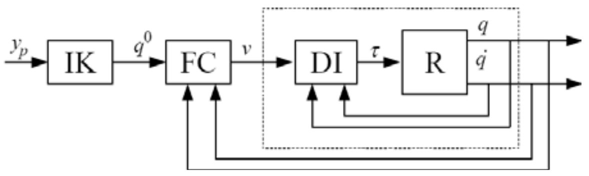


Figure 4. Block diagram of inverse dynamics control.

velocities q and $\dot{q}$ (represented by the dotted block) is now described by a simple integrator and a double integrator, respectively. It is then easy to tune a fixed-parameter, linear feedback controller (FC) in order to obtain the desired closed-loop dynamics.

Starting from the index-1 DAE robot model (20), it is straightforward to derive the equations and then the explicit algorithms to compute the DK, IK, CT, and DI, by using the same techniques employed to bring (20) into state-space form. The DK (17)-(18) is obtained by solving (20) for y_p (and possibly y_v) given q (and possibly $\dot{q}$), while the IK is obtained by solving (20) for q (and possibly $\dot{q}$) given y_p (and possibly y_v); the subset of required equations is found by suitable analysis of the incidence matrix. The CT (25) is obtained by solving (20) for τ given q, $\dot{q}$, and $\ddot{q}$. Finally, the DI (26) is obtained by solving (20) augmented with (26) τ given v, q, and $\dot{q}$. EOOL tools should then be able to automatically generate the code corresponding to the DK, IK, CT, and DI blocks in two forms: as algorithms to compute the outputs given the inputs (e.g., C code for direct inclusion in the robot controller), as well as equationbased Modelica blocks, which could be used for closedloop simulation within a Modelica environment.

As a final remark, note that the method of inverse dynamics is a special case of the much more general theory of feedback linearization [20], whose goal is to obtain a LTI dynamics made by pure integrators from generic nonlinear systems, by applying suitable feedback actions as shown in Figure 4. It could also be interesting to investigate the coupling between EOOL tools and symbolic manipulation tools for the design of such controllers.

4.4 Fast and compact models for model predictive control

The Model Predictive Control (MPC) approach [25, 36] is based on a few key ideas, that turn the control problem into an optimization problem. The control variable is a discretetime variable, that changes periodically every T_s seconds:

$$u(t) = u(k), \quad kT_s \le t < (k+1)T_s \tag{28}$$

At each time step k, an optimization problem is solved, whose unknowns are the next values of the control variable $u(k+i)$ over a finite horizon $1 \le i \le n$. The first sample $u(k+1)$ is then applied to the actuators at the next time step, the rest of the values are discarded, and the process is repeated over and over, thus implementing a *receding horizon strategy*.

There are different ways to formulate the MPC problem, depending on the specific technique used to solve the problem. Generally speaking, the figure of merit to be minimized is a quadratic function, which suitably weights the future deviations of the controlled variables from the set point and the intensity of the control action, as well as any other problem-specific performance index that has to be minimized, e.g. the financial cost of running the process. The constraints of the optimization problem are the dynamic relationship between the input and output variables, typically in the form (4), and possibly other constraints, such as upper and lower bounds of the state, control and output variables and of their rate of change.

The main advantage of MPC is its intrinsic ability to deal with highly interacting multivariable systems (many control inputs and controlled outputs), while keeping into account operating constraints such as actuator saturations or hard bounds on controlled variables, and at the same time meeting some problem-specific optimality criterion. The main drawback is the high computational load, since a (possibly nonlinear and non-convex) constrained optimization problem must be solved at each sampling time; this makes MPC suitable for systems with slow dynamics, e. g. chemical plants, where there is plenty of time to carry out the required computations in real time. This limitation is likely to become less and less stringent in the future, thanks to Moore's law.

The second issue is the requirement that a suitable plant model is available, as the control system performance critically depends on the model quality. Models for linear MPC can be obtained either by linearization of analytical models, or by system identification from experimental and/or simulation data, e. g. step responses; both cases are already supported by current EOOL tools. Nonlinear MPC (NMPC) algorithms are preferably based upon analytical models in state-space form (1), which are derived from physical firstprinciples models. The conversion to discrete-time form (4) is often performed internally by the

NMPC algorithm itself, by standard ODE integration routines. This means that the interface between the EOOL tool and the NMPC tool is similar to the one used for simulation problems, i.e. the state-space form (1), possibly augmented by the Jacobians of the right-hand-sides of (1).

The main requirement for NMPC-oriented models is that they must have the least possible number of state and algebraic variables, in order to keep the complexity of the optimization problem within acceptable limits, and that they have good smoothness properties, in order to avoid convergence problems of the iterative optimization algorithms. The development of those models can be very time consuming, and require highly skilled manpower; it is apparent how better tool support could be extremely useful in order to reduce the development effort and cost.

The potential of OOM for MPC was first noted by Maciejowski at the end of the '90 [24]. There are several reported case studies [14, 15, 3, 19], where the model used in the NMPC algorithm was derived from a Modelica model of the physical plant, using the tool Dymola to produce the code corresponding to the state-space form (1), i.e., the *dsmodel.c* code that is usually linked to ODE/DAE solvers. In order to derive suitably simplified models, the features of Modelica discussed in Section 3.3 have been extensively exploited. In general, this approach has proven much more satisfactory than writing the C-code of the model from scratch; however, it still requires a substantial investment of time and effort for each new application.

The application of the automatic MOR techniques described in section 4, possibly still combined with some manual intervention in terms of replaceable models, looks very promising in order to bring detailed simulation models into a form which is suitable for NMPC with a much more limited effort by the developer.

Furthermore, [19] correctly points out that, although the interface to NMPC algorithms is very similar to the interface to ODE/DAE solvers, the former requires some more flexibility. For example, advanced NMPC schemes can provide on-line estimation of uncertain parameters through the use of extended or unscented Kalman filters. This means that some model parameters are no longer constant throughout a transient, so that the C-code obtained for simulation purposes must be manually adapted. A better option would be to implement a code export interface which makes it possible to turn selected parameters appearing in (5) (which are going to be estimated on-line) into inputs, before transforming the system in state-space form (1).

5 Conclusions

After a brief review of the different uses of models in control system design, the current state of the art of EOOL-based tools for CACSD has been reviewed: apparently, currently available tools mainly focus on simulation tasks. Several further directions for research and development in EOOL tools where then discussed, which go beyond the mere simulation problem. Results in these directions could substantially improve the level of support to the control engineer willing to apply advanced, model-based control techniques to real-life problems, starting from object-oriented models of the plant.

References

[1] OpenModelica home page. URL: http://www.ida. liu.se/labs/pelab/modelica/OpenModelica. html.

[2] J. Åkesson. *Optimica - An extension of Modelica supporting dynamic optimization*. In Proc. 6th Int. Modelica Conference, pages 57–66, Bielefeld, Germany, Mar. 3–4 2008.

[3] J. Åkesson O. Slätteke. *Modeling, calibration and control of a paper machine dryer section*. In Proc. 5th Int. Modelica Conference, pages 411–420, Vienna, Austria, Sep. 4–5 2006.

[4] M. Andersson, S. E. Mattsson, D. Brück, T. Schöntal. *OmSim - an integrated environment for object-oriented modelling and simulation*. In Proc. IEEE/IFAC Joint Symposium on Computer-Aided Control System Design, CACSD'94, pages 285–290, Tucson, Arizona, March 1994.

[5] P. Apkarian, R. J. Adams. *Advanced Gain-Scheduling Techniques for Uncertain Systems*. IEEE Transactions on Control System Technology, 6:21–32, 1998.

[6] P. I. Barton, C. C. Pantelides. *The modeling of combined continuous and discrete processes*. AIChE Journal, 40:966– 979, 1994.

[7] D.A. van Beek, M.A. Reniers, J.E. Rooda, R.R.H. Schiffelers. *Foundations of an interchange format for hybrid systems*. In A. Bemporad, A. Bicchi, G. Butazzo, eds., Computation and Control, 10th International Workshop, volume 4416 of Lecture Notes in Computer Science, pages 587–600. Springer Verlag, 2007.

[8] D.A. van Beek, M.A. Reniers, J.E. Rooda, R.R.H. Schiffelers. *Concrete syntax and semantics of the compositional interchange format for hybrid systems*. In Proc. 17th IFAC World Congress, Seoul, Korea, Jul 6–11 2008.

[9] E. Carpanzano, C. Maffezzoni. *Symbolic manipulation*

techniques for model simplification in object-oriented modelling of large scale continuous systems. Mathematics and Computers in Simulation, 48(2):133–150, 1998.

[10] F. E. Cellier, H. Elmqvist. *Automated formula manipulation supports object-oriented continuous-system modeling*. IEEE Control Systems Magazine, 13(2):28–38, 1993.

[11] H. Elmqvist, S. E. Mattsson, H. Olsson. *New methods for hardware-in-the-loop simulation of stiff models*. In Proc. 2nd International Modelica Conference, pages 59–64, Oberpfaffenhofen, Germany, Mar. 18–19 2002.

[12] H. Elmqvist, M. Otter, F. Cellier. *Inline integration: A new mixed symbolic /numeric approach for solving differential–algebraic equation systems*. In Proc. ESM'95, European Simulation Multiconference, pages xxiii–xxxiv, Prague, Czech Republic, Jun. 5–8 1995.

[13] The Fraunhofer-Institut für Techno-und Wirtschaftsmathematik. *Analog Insydes*. URL: http://www. analog-insydes.de/.

[14] R. Franke. *Formulation of dynamic optimization problems using modelica and their efficient solution*. In Proc. 2nd International Modelica Conference, pages 315–323, Oberpfaffenhofen, Germany, Mar. 18-19 2002.

[15] R. Franke, M. Rode, K. Krüger. *On-line optimization of drum boiler startup*. In Proc. 3rd International Modelica Conference, pages 287–296, Nov. 3–4 2003.

[16] P. Fritzson, P. Aronsson, A. Pop, H. Lundvall, K. Nyström, L. Saldamli, D. Broman, A. Sandholm. *Open-Modelica A free open-source environment for system modeling, simulation, and teaching*. In Proceedings IEEE International Symposium on Computer-Aided Control Systems Design, Munich, Germany, Oct. 4–6 2006.

[17] T. Halfmann, T. Wichmann. *Symbolic methods in industrial analog circuit design*. In Scientific Computing in Electrical Engineering, Mathematics in Industry. Springer Verlag, 2006.

[18] S. Hecker, A. Varga. *Symbolic manipulation techniques for low order LFT-based parametric uncertainty modelling*. International Journal of Control, 79(11):1485–1494, 2006.

[19] L. Imsland, P. Kittilsen, T. S. Schei. *Model-based optimizing control and estimation using modelica models*. In Proc. 6th International Modelica Conference, pages 301–310, Bielefeld, Germany, Mar. 3–4 2008.

[20] H. K. Khalil. *Nonlinear Systems*. Prentice Hall, 3rd edition, 2002.

[21] L. Lee, K. Poolla. *Identification of linear parametervarying systems using nonlinear programming*. Journal of Dynamic Systems, Measurement and Control - Transactions of the ASME, 121(1):71–78, 1999.

[22] D. J. Leith, W. E. Leithead. *Survey of gain-scheduling analysis and design*. International Journal of Control, 73(11):1001–1025, 2000.

[23] M. Lovera, G. Mercere. *Identification for gainscheduling: a balanced subspace approach*. In 2007 American Control Conference, New York, USA, 2007.

[24] J. M. Maciejowski. *Modelling and predictive control: enabling technologies for reconfiguration*. Annual Reviews in Control, 23(1):13–23, 1999.

[25] J. M. Maciejowski. *Predictive control: with constraints*. Prentice Hall, 2002.

[26] C. Maffezzoni, R. Girelli. *Modular modelling in an object-oriented database*. Mathematical Modelling of Systems, 4:121–147, 1998.

[27] J.-F. Magni. *Linear fractional representation toolbox*. Technical Report TR 6/08162 DCSD, ONERA, 2004.

[28] Marcos, G. Balas. *Development of linear-parametervarying models for aircraft*. Journal of Guidance, Control and Dynamics, 27(2):218–228, 2004.

[29] S. E. Mattsson, M. Andersson, K. J. Åström. *Modeling and simulation of behavioral systems*. In Proc. 32nd IEEE Conference on Decision and Control, volume 4, pages 3636–3641, San Antonio, Texas, Dec. 1993.

[30] S. E. Mattsson, M. Andersson, K. J. Åström. *Objectoriented modelling and simulation*. In D. Linkens, ed., CAD for Control Systems, pages 31–69. Marcel Dekker Inc., New York, 1993.

[31] S. E. Mattsson, H. Elmqvist. *Simulator for dynamical systems using graphics and equations for modeling*. IEEE Control Systems Magazine, 9(1):53–58, Jan. 1989.

[32] S. E. Mattsson, H. Elmqvist, M. Otter. *Physical system 44 modeling with Modelica*. Control Engineering Practice, 6(4):501–510, 1998.

[33] M. Otter, H. Elmqvist, S. E. Mattsson. *The new Modelica MultiBody library*. In Proc. 3rd International Modelica Conference, pages 311–330, Linköping, Sweden, Nov. 3–4 2003. URL: http://www.modelica. org/events/Conference2003/papers/h37_Otter_multi-body.pdf.

[34] Paijmans,W. Symens, H. Van Brussel, and J. Swevers. *A gain-scheduling-control technique for mechatronic systems with position-dependent dynamics*. In Proc. 2006 American Control Conference, Minneapolis, USA, 2006.

[35] Pop, P. Fritzson. *ModelicaXML: A Modelica XML representation with applications*. In Proc. 3rd International Modelica Conference, pages 419–430, Linköping, Nov 3–4 2003.

[36] S. J. Qin, T. A. Badgwell. *A survey of industrial model predictive control technology*. Control Engineering Practice, 11:733–764, 2003.

[37] W. Rugh, J. Shamma. *Research on gain scheduling*. Automatica, 36(10):1401–1425, 2000.

[38] P. Schwarz, J. Bastians, C. Clauss, J. Haase, A. Köhler, G. Otte, P. Schneider. *A tool-box approach to computeraided generation of reduced-order models*. In Proc. EUROSIM 2007, 2007.

[39] L. Sciavicco, B. Siciliano. *Modelling and Control of Robot Manipulators*. Springer Verlag, 2000.

[40] R. Sommer, T. Halfmann, J. Broz. *Automated behavioral modeling and analytical model-order reduction by application of symbolic circuit analysis for multiphysical systems*. In Proc. EUROSIM 2007, 2007.

[41] The Modelica Association. *Modelica - A unified objectoriented language for physical systems modeling - Lan-*

guage specification version 3.0. Online, Sep. 5 2007. URL: http://www.modelica.org/news_items/ documents/ModelicaSpec30.pdf.

[42] The World Wide Web Consortium. *Mathematical Markup Language (MathML) Version 2.0*. Online, Oct 21 2003. URL: http://www.w3.org/TR/MathML2/.

[43] J.J.M. van Helvoort, M. Steinbuch, P.F. Lambrechts, R. van de Molengraft. *Analytical and experimental modelling for gain-scheduling of a double scara robot*. In Proc. 3rd IFAC Symposium on Mechatronic Systems, Sydney, Australia, 2004.

[44] Varga, G. Looye, D. Moormann, G. Gräbel. *Automated generation of LFT-based parametric uncertainty descriptions from generic aircraft models*. Mathematical and Computer Modelling of Dynamical Systems, 4(4):249–274, 1988.

[45] V. Verdult. *Nonlinear system identification: a state space approach*. PhD thesis, University of Twente, 2002.

[46] K. Zhou, J. U. Doyle, and K. Glover. Robust and optimal control. Prentice-Hall, New Jersey, 1996.

Corresponding author: Francesco Casella
Politecnico di Milano
Dipartimento di Elettronica e Informazione, Italy
casella@elet.polimi.it

Accepted EOOLT 2008, June 2008
Received: September 17, 2008
Accepted: January 14, 2009

SHORT NOTES

Implementation of a Distributed Consensus Algorithm with OMNeT++

Andreas Dielacher, Thomas Handl, Christian Widtmann, Vienna University of Technology, Austria
{dielacher, handl, widtmann}@ecs.tuwien.ac.at

SNE Simulation Notes Europe SNE 19(1), 2009, 41-42, doi: 10.11128/sne.19.sn.09927

This paper examines the implementation of a simple distributed consensus algorithm with OMNET++. It will be shown that using this simulator *(i)* comes at nearly no cost and *(ii)* massively improves comprehension of the system under study.

Introduction

One of the major concerns of distributed computing is the ability of a group of nodes or processors to agree on a common value. This agreement is also called (distributed) consensus. Furthermore, in a realistic system model it is necessary to allow nodes to fail. The nature of failure depends on the fault model chosen. In the benign case, a node fails to send its messages in a consistent way (e.g. crash failures), whereas in the malicious case one has to deal with byzantine behavior. In the following we will focus on consensus with crash failures. This paper is structured as follows. Section 1 deals with the chosen simulator. We then describe the problem in Section 2 and our implementation in Section 3. The paper concludes in Section 4 with our results.

1 OMNeT++

OMNeT++ [2] is a modular open-source simulation environment with GUI support. It provides communication primitives allowing to easily model communication networks and alike although it has already been used in other areas like hardware architectures and business processes. This tool is used for scientific research as well as for industrial engineering. Examples for open source simulation models are network protocols like IP, IPv6 or MPLS.

From a technical point of view, OMNeT++ is a discrete and event driven simulator generator. The behavior of the system to be simulated is modeled using the well known C++ programming language. The structure of this system is described in a proprietary language called NED. OMNeT++ then compiles this code into a stand-alone simulator with GUI.

OMNeT++ offers many features such as user-defined message definition, message statistics, message tracking, visualization of network traffic and many more. Nevertheless, there are some inconveniences to handle. For example it is not possible to implement timeouts directly. This deficiency has to be overcome by using "self messages", a usual approach in distributed computing models.

```
Initially $V = \{x\}$.
for round $k, 1 \le k \le f+1$ do
    send $v \in V : p_i$ has not already sent $v$ to all proc.
    receive $S_j$ from $p_j$, $0 \le j \le n-1$, $j \ne i$
    $V := V \cup \bigcup_{j=0}^{n-1} S_j$
    if $k = f+1$ then $y = \min(V)$
```

Algorithm 1. Simple consensus, code for one processor p_i

2 Distributed consensus with crash failures

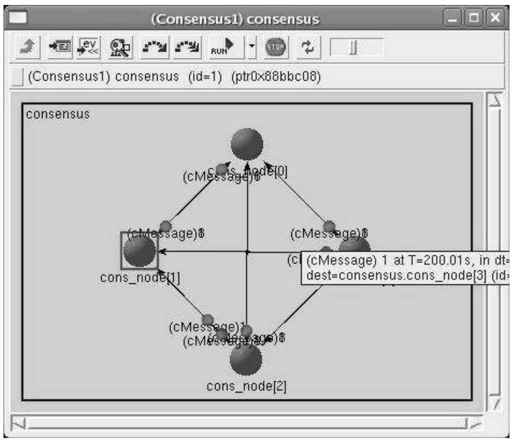
In this paper we focus on the standard synchronous model, where processors act in a round-based manner. Each round consists of sending and receiving one or more messages and one zero time computing step on each processor. Computation steps occur at the beginning of each round.

The algorithm we implemented is taken from [1] and listed in Algorithm 1. Each processor has a—probably distinct—value x. It is the intention of this algorithm to provide every processor with the same value after its completion. This indicates that nodes have to share their data and need common criteria to decide it. The algorithm operates such that in every round each processor sends a value it has not sent yet to all other processors. Each processor stores the received values in its set V. We allow one processor per round to fail with crash failure. In particular, we allow a failing node to send an arbitrary number of correct messages in the specific round it fails. After the crash a processor is not allowed to send any more messages. For this algorithm to work, the network has to be fully connected. It can be shown that the algorithm needs at least $f+1$ nodes, where f is the amount of crash failures tolerated.

3 Implementation

Knowledge of the C++ programming language can usually be presumed. So the main challenge in implementing a distributed algorithm in software usually turns out to be the comprehension that Algorithm 1 describes the behavior of one instance of a node whereas the system under study constitutes many nodes processing concurrently.

As shown in Figure 1 the GUI of the simulator created by OMNeT++ is quite intuitive. It can be seen that our model is made up of four nodes, thus tolerating three crash failures. Besides simulating the behavior of a fault-free net in the first place, we also deployed crashing nodes. In order to save compile time, we exploited a certain feature of OMNeT++: parameterization of the modeled system. By providing the



(Consensus1) consensus (id=1) (ptr0x88bbc08)

(cMessage) 1 at T=200.01s, in dt=
dest=consensus.cons_node[3] (id=

Figure 1. GUI of OMNeT++

number of nodes to crash as a parameter, arbitrary experiments can be conducted easily.

The implementation of the behavior of this algorithm requires about 150 lines of code.

4 Conclusion

In this paper, we presented the successful implementation of a simple but nevertheless important building block of distributed computing: a consensus algorithm. In more detail, we have shown that the chosen simulator, OMNeT++, is *(i)* capable of simulating such an algorithm and *(ii)* particularly useful for the purpose of demonstration and also education. OMNeT++ has already proven to be very useful for the distributed computing community.

In a further step, we will extend the fault model to arbitrary (byzantine) faults and also examine data fusion algorithms. The results are intended to establish the basis for a hardware implementation of such an algorithm serving as a fault tolerance layer in distributed systems.

References

[47] H. Attiya, J. Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics.* Wiley Series on Parallel and Distributed Computing. Wiley and Sons, 2004.

[48] OMNeT++ Community Site. http://www.omnetpp.org

Corresponding author: Andreas Dielacher,
Department of Computer Engineering
Vienna University of Technology
Wiedner Hauptstraße 8-10, 1040 Vienna, Austria
dielacher@ecs.tuwien.ac.at

Received: October 12, 2008
Revised: January 20, 2009
Accepted: February 5, 2009

Four Modelling Approaches for ARGESIM Benchmark C3 'Generalized Class-E Amplifier' implemented in Dymola

Gemma Ferdinand Kaunang, Günther Zauner, Vienna University of Technology, Austria

SNE Simulation Notes Europe SNE 19(1), 2009, 43-46, doi: 10.11128/sne.19.bn03 .09929

Simulator: Dymola—**Dy**namic **Mo**deling **La**boratory—is suitable for modeling of various kinds of physical systems and the combination of systems of different domains. It supports hierarchical model composition, libraries of truly reusable components, connectors and composite acasual connections. Model libraries are available in many engineering domains.

Dymola has a powerful graphic editor for composing models. Dymola is based on the use of Modelica models stored on files.

Dymola has powerful experimentation, plotting and animation features. Scripts can be used to manage experiments and to perform calculations. Automatic documentation generator is provided.

Model: The basic class-E power amplifier was introduced by N.O. Sokal and A.D. Sokal in their classic paper from 1975. It is a switching-mode amplifier that operates with zero voltage and zero slope across the switch at switch turn-off. The actual numerical example is taken from J.C. Mandojana, K.J. Herman and R.E. Zulinski. They use the following equivalent circuit of a generalized class-E amplifier as a test example for a procedure to evaluate steady state boundary conditions by means of MATLAB. Figure 1 shows Class-E Amplifier: The component values are: VDC= 5 V, L1 = 79.9 µH, C2 = 17.9 nF, L3 = 232 µH, C4 = 9.66 µF and RL = 52.4Ω.

The time dependent resistor $R(t)$ models the active device acting as a switch with an ON-resistance of 0.05Ω and an OFF-resistance of $5M\Omega$. An extreme ON-resistance of value zero ohm will of course result in a pathological system i.e. the old story of what happens when an ideal capacitor with a certain charge is suddenly short circuited. Furthermore the DC voltage source will be short circuited through the ideal coil $L1$. Figure 2 shows function of time $R(t)$. The duty ratio is 50%. The period is $10\mu s$ (frequency $100kHz$). The rise/fall time TRF is $1fs$.

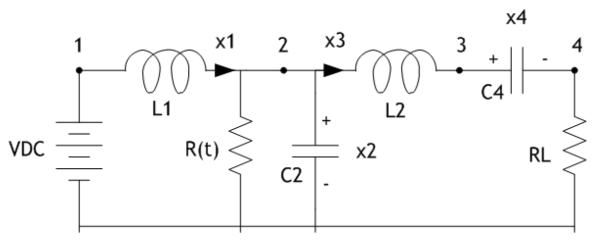


Figure 1: Class-E amplifier

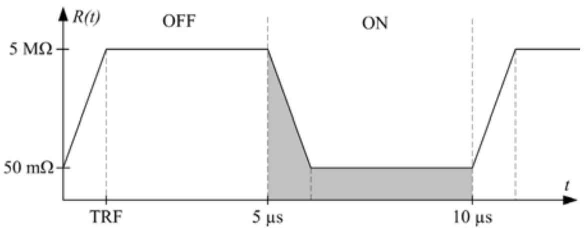


Figure 2. Time dependent resistor $R(t)$, not to scale.

The equations describing the circuit may be the state-equations where inductor currents and capacitor voltages are chosen as system variables. By using the Kirchhoff voltage and current laws we get the following differential equations:

$$dx1/dt = (-x2 + VDC)/L1$$

$$dx2/dt = (x1 - x2/R(t) - x3)/C2$$

$$dx3/dt = (x2 - RL * x3 - x4)/L3$$

$$dx4/dt = x3/C4$$

The aim was to implement these equations into Dymola structure. Therefore four different ways has been chosen to model these equations.

The first solution is by using modelica text language. Designing the model is relative easy in modelica by using the exact equation above in the equation section. The time dependent resistor is modelled in the `algorithm` section and the whole modeling code has the following structure:

```
model C3Dymola_textv2
  constant Real L1 = 79.9E-6;
  constant Real C2 = 17.9E-9;
  ...     // The same for the other values
  Real x1, x2, x3, x4;
  Real Rt;
```

```
   Real t_red;
   Real IRT;
   Real VRL;
   Real k;
   ...
   equation
     t_red = mod(time, 10E-6);
     k = ((5e+6)-(5e-2))/TRF;
     ...
   algorithm
      if (0<=t_red) and (t_red<TRF)
         then Rt:=(5e-2) + k*t_red;
      elseif (TRF<=t_red) and (t_red<(5e-6))
         then Rt:=5e+6;
      elseif ((5e-6)<=t_red) and
                   (t_red<((5e-6)+TRF))
         then Rt:=(5e+6) - k*(t_red - (5e-6));
      elseif ((5e-6)+TRF<=t_red) and
                  (t_red<(10e-6))
         then Rt:=5e-2;
         else Rt:=-5;
      end if;
   equation
      L1 * der(x1)= -x2 + VDC;
      C2 * der(x2)= x1 - (x2/Rt) - x3;
      L3 * der(x3)= x2 - (RL*x3) - x4;
      C4 * der(x4)= x3;
end C3Dymola_textv2;
```

Listing 1: C3 basic model implementation in Dymola Text mode

The second solution is using a block diagram model. The differential equations above [(1) to (4)] were built by block integrator, add/subtract and gain, also a division block for $x2/R(t)$ and constant block for VDC. The time dependent resistor is built by trapezoid source block by defining its parameters as follows:

Amplitude	5e+6	Offset	5e−2
Rising	1e−15	Falling	1e−15
Width	5e−6	Period	10e−6

The trapezoid source block is shown by Figure 3. The model of the system block in the diagram layer is shown in Figure 4.

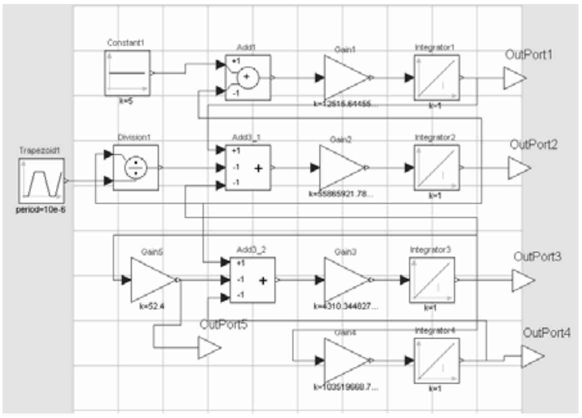


Figure 4. Model of the Differential Equation System in block style (block diagram)

The third solution is by using a state graph model. Designing the model based on second solution by adding a triggered trapezoid block as time dependent resistor by defining its parameter as follows:

Amplitude	5e+6	Offset	5e−2
Rising	1e−15	Falling	1e−15

The triggered trapezoid block used in the third modeling approach is shown in Figure 5.

Also by adding greater equal, less equal block and stategraph model block such as initial step, step and transition to activate triggered trapezoid from input trapezoid source block, which have the same parameter from second solution. Greater equal and less equal block will determine what state will be activated in stategraph model, is it state off (when $R(t)=5M\Omega$) or state on (when $R(t)=5m\Omega$). Basically the stategraph model consists only of 1 initial step, 2 transition blocks and 1 step block. The initial step block will represent state off and step block will represent state on. State off will change to state on whenever transition block 1 active, which being contrrolled by output from less equal block and state on will change to state on whenever transition block 2 active, which being

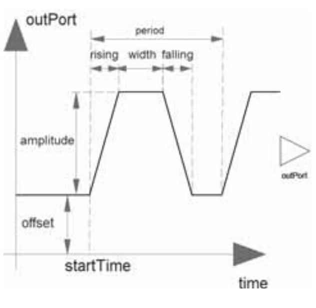


Figure 3. Trapezoid source block.

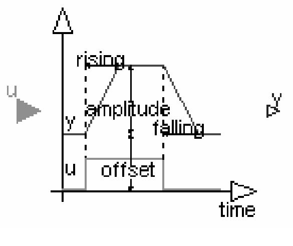


Figure 5: Triggered Trapezoid Block with input u and output y, whereby u is a logical variable and y is of type Real.

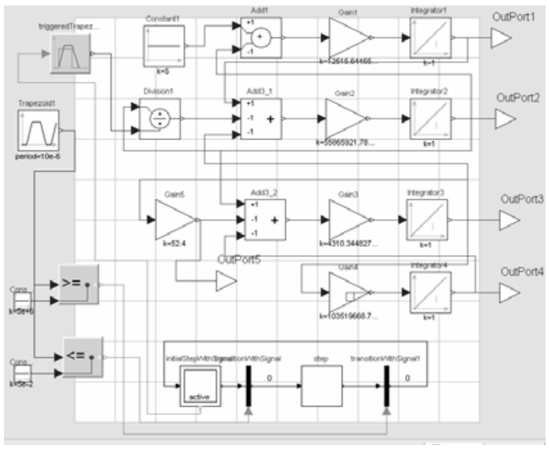


Figure 6. Model of the System (stategraph)

controlled by output from greater equal block. Whenever initial step active, it will send trigger signal to activate triggered trapezoid block, which will have output as a trapezoid signal. The model of the system for stategraph model is displayed in Figure 6.

The fourth solution is using the electrical model equivalent circuit. Designing the model using components from the Modelica standard library `Modelica.Electrical.Analog.Basic`, like resistor, inductor, capacitor, ground and constant voltage. The time dependent resistor $R(t)$ was built by using the component `VariableResistor` from the electrical library and trapezoid source defined in the block library, which has the same parameters as defined in second solution.as input of variable resistor. The model of the system for electrical model is shown in Figure 7.

All block models are built in the `Diagram` layer of the Dymola modeling interface without using the `Equation` layer. Setting up the simulation parameters, simulation process and plotting the result are done in `Simulation window` of Dymola.

A-Task: To calculate the eigenvalues of the system when $R(t) = 50\,\text{m}\Omega$ and in case when $R(t) = 5\,\text{M}\Omega$ is done by calling the function `eigenValues` (included in Modelica standard library 2.2) and putting the values of all variables into a matrix syste. The matrix system for Task A is defined as follows:

$$\begin{pmatrix} 0 & -1/L1 & 0 & 0 \\ 1/C2 & -1/[R(t)C2] & -1/C2 & 0 \\ 0 & 1/L3 & -RL/L3 & -1/L3 \\ 0 & 0 & 1/C4 & 0 \end{pmatrix}$$

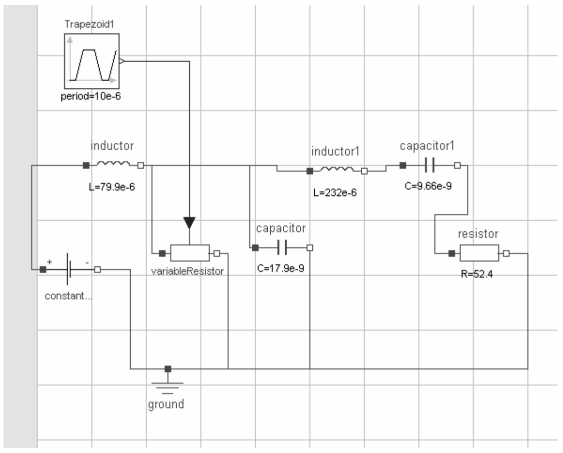


Figure 7: Electrical model representation of the Class-E Amplifier as defined in ARGESIM Benchmark 3 in the `Diagram` layer of Dymola.

ON period	OFF period
−1,1303e+005 + 6,5835e+005i	−5,4708 e+004 + 1,0407e+006i
−1,1303e+005 – 6,5835e+005i	−5,4708 e+004 - 1,0407e+006i
−6,258e+002	−5,8228e+004 + 5,3275e+005i
−1, 117e+009	−5,8228e+004 - 5,3275e+005i

Table 1: Eigenvalues of $R(t)$ in the ON – and OFF – periode of the system

It took 0.5 sec to execute Task A for all solutions. The result of eigenvalue analysis is shown in Table1 for all solution.

B-Task: To simulate the system in all four implementations, the Dassl integration method was used (settings: 1000 as number of intervals, 0…100μs as simulation time interval and 1e-4 as a relative tolerance). Under the initial value zero, for $x1, x2, x3$ and $x4$, the result for variable current switch resistor $IR(t)$ and output voltage VRL is given by figure 8. It took 0,047s, 0,047s, 0,063s and 0,047s to simulate the Task B for 1st, 2nd, 3rd and 4th solution respectively.

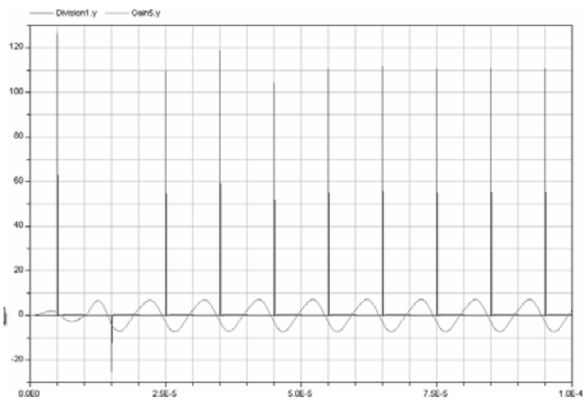


Figure 8: Time Curve $IR(t)$ and VRL

C-Task: The parameter of *TRF* is varied between $1fs, 10ps, 1ns$ and $100ns$. Initial value for Task C is the final solution given by Task B. The time interval is $0...9\mu s$. Changing the time curve plot into phase plot works by one mouse click, choosing independent variable instead of time. As result, the phase plane curves $dx3/dt = VL3$ as a function of $x3 = IL3$ are given by Figure 9. It took 0,031s, 0,025s, 0,047s and 0,015s to simulate Task C for 1st, 2nd, 3rd and 4th solution respectively.

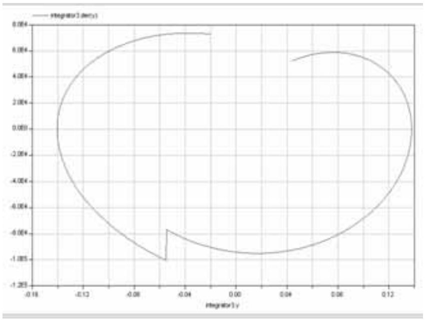
Resumé: Dymola is a powerful modeling and simulation tool that aids implemrntation of ordinary differential systems in different ways. By calling eigenvalue function for doing Task A it overs a simple way for additional system analysis. Using right mouse click to change between plots based on time and plots based on independent variable uses the strength of the included plotting and representation tool. Comparing to other component based simulation tool Dymola works very fast. Each of the tasks is calculated in less then $0.51s$ on a standard PC.

The ranking of solutions based on simulation time from fast to slow is:

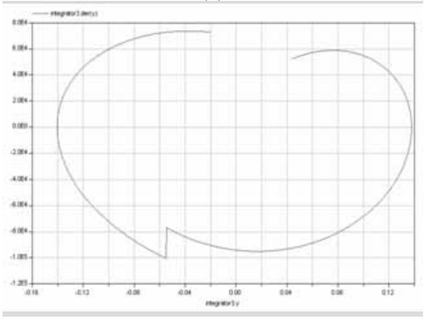
1. Textual mode
2. Electrical model
3. Block Diagram model
4. Stategraph model

Corresponding author: Gemma Ferdinand Kaunang,
Department of Analysis and Scientific Computing
Vienna University of Technology
Wiedner Hauptstraße 8-10, 1040 Vienna, Austria
ferdiv_555@yahoo.de

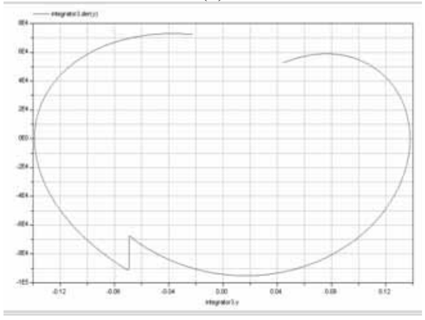
Received: August 17, 2008
Revised: October 10, 2008
Accepted: January 12, 2009



(a)



(c)



(d)

Figure 9. Phase plane curve VL3 function IL3 for TRF (a) 1e-15s (b) 1e-11s (c) 1e-9s (d) 1e-7s

SNE News Section

EUROSIM Data & Quick Info

CZECH TECHNICAL UNIVERSITY IN PRAGUE

EUROSIM 2010

organised by CSSS

September 5 – 9, 2010, Prague, Czech Republic

ES

Contents

Simulation News Europe is the official journal of EUROSIM and sent to most members of the EUROSIM Societies as part of the membership benefits. Furthermore **SNE** is distributed to other societies and to individuals active in the area of modelling and simulation. SNE is registered with ISSN 1015-8685. Circulation of printed version is 3000 copies.

eSNE – SNE at Web SNE issues are also available at `www.argesim.org` as eSNE – *Electronic SNE* . Web-resolution eSNEs are free for download. Subscribers, e.g. members of EUROSIM Societies have access to SNE Archive with high-resolution eSNE copies and and with sources of ARGESIM Benchmarks.

This *EUROSIM Data & Quick Info* compiles data from EUROSIM and EUROSIM societies: addresses, weblinks, officers of societies with function and email, to be published regularly in SNE issues – independent of individual reports of the societies.

SNE Reports Editorial Board

→ `www.sne-journal.org`
✉ `office@sne-journal.org`
→ `www.eurosim.info`

If you have any information, announcement, etc. you want to see published, please contact a member of the editorial board in your country or to office@sne-journal.org.

Editorial Information/Impressum - see front cover

Information EUROSIM

EUROSIM Federation of European Simulation Societies

General Information. *EUROSIM*, the Federation of European Simulation Societies, was set up in 1989. The purpose of EUROSIM is to provide a European forum for regional and national simulation societies to promote the advancement of modelling and simulation in industry, research, and development.

→ www.eurosim.info

Member Societies. EUROSIM members may be national simulation societies and regional or international societies and groups dealing with modelling and simulation. At present EUROSIM has thirteen full members and one observer member:

ASIM	Arbeitsgemeinschaft Simulation *Austria, Germany, Switzerland*
CEA-SMSG	Spanish Modelling and Simulation Group *Spain*
CROSSIM	Croatian Society for Simulation Modeling *Croatia*
CSSS	Czech and Slovak Simulation Society *Czech Republic, Slovak Republic*
DBSS	Dutch Benelux Simulation Society *Belgium, Netherlands*
FRANCOSIM	Société Francophone de Simulation *Belgium, France*
HSS	Hungarian Simulation Society *Hungary*
ISCS	Italian Society for Computer Simulation *Italy*
LSS	Latvian Simulation Society *Latvia*
PSCS	Polish Society for Computer Simulation *Poland*
SIMS	Simulation Society of Scandinavia *Denmark, Finland, Norway, Sweden*
SLOSIM	Slovenian Simulation Society *Slovenia*
UKSIM	United Kingdom Simulation Society *UK, Ireland*
ROMSIM	Romanian Society for Modelling and Simulation, *Romania, Observer Member*

Contact addresses, weblinks and officers of the societies may be found in the information part of the societies.

EUROSIM board/EUROSIM officers. EUROSIM is governed by a board consisting of one representative of each member society, president and past president, and representatives for SNE and SIMPRA. The President is nominated by the society organising the next EUROSIM Congress. Secretary and Treasurer are elected out of members of the Board.

President	Mikuláš Alexík (CSSS), *alexik@frtk.fri.utc.sk*
Past president	Borut Zupančič (SLOSIM) *borut.zupancic@fe.uni-lj.si*
Secretary	Peter Fritzson (SIMS) *petfr@ida.liu.se*
Treasurer	Felix Breitenecker (ASIM) *felix.breitenecker@tuwien.ac.at*
SIMPRA Repres.	Jürgen Halin *halin@iet.mavt.ethz.ch*
SNE Repres.	Felix Breitenecker *felix.breitenecker@tuwien.ac.at*

SNE – Simulation News Europe. EUROSIM societies are offered to distribute to their members the journal *Simulation News Europe* (SNE) as official membership journal. SNE is a scientific journal with reviewed contributions in the *Notes Section* as well as a membership newsletter for EUROSIM with information from the societies in the *News Section*. Publisher are EUROSIM, ARGESIM and ASIM.

Editor-in-chief	Felix Breitenecker *felix.breitenecker@tuwien.ac.at*

→ www.sne-journal.org, menu SNE
🖃 office@sne-journal.org

EUROSIM Congress. EUROSIM is running the triennial conference series EUROSIM Congress. The congress is organised by one of the EUROSIM societies. EUROSIM 2010 will be organised by CSSS in Prague, September 5-10, 2010.

Organisation EUROSIM 2010	Miroslav Šnorek *snorek@fel.cvut.cz*
Information CSSS	Mikulas Alexik (CSSS) *alexik@frtk.utc.sk*

→ www.eurosim2010.org

ASIM German Simulation Society Arbeitsgemeinschaft Simulation

ASIM (Arbeitsgemeinschaft Simulation) is the association for simulation in the German speaking area, servicing mainly Germany, Switzerland and Austria. ASIM was founded in 1981 and has now about 700 individual members, and 30 institutional or industrial members. Furthermore, ASIM counts about 300 affiliated members.

→ www.asim-gi.org with members' area

info@asim-gi.org, admin@asim-gi.org

ASIM – Inst. f. Analysis and Scientific Computing
Vienna University of Technology
Wiedner Hauptstraße 8-10, 1040 Vienna, Austria

ASIM Officers

President	Felix Breitenecker *felix.breitenecker@tuwien.ac.at*
Vice presidents	Sigrid Wenzel *s.wenzel@uni-kassel.de*
	Thorsten Pawletta, *pawel@mb.hs-wismar.de*
Secretary	Anna Mathe, *anna.mathe@tuwien.ac.at*
Treasurer	Ingrid Bausch-Gall, *Ingrid@Bausch-Gall.de*
Membership affairs	S. Wenzel, *s.wenzel@uni-kassel.de*
	W. Maurer, *werner.maurer@zhwin.ch*
	I. Bausch-Gall, *Ingrid@Bausch-Gall.de*
	F. Breitenecker (*mail address above*)
Universities	W. Wiechert *wiechert@simtec.mb.uni-siegen.de*
Industry	S. Wenzel, *s.wenzel@uni-kassel.de*
	K. Panreck, *Klaus.Panreck@hella.com*
Conferences	Klaus Panreck *Klaus.Panreck@hella.com*
	Albrecht Gnauck *albrecht.gnauck@tu-cottbus.de*
Publications	Th. Pawletta, *pawel@mb.hs-wismar.de*
	F. Breitenecker (*mail address above*)
Repr. EUROSIM	F. Breitenecker (*mail address above*)
Deputy	W. Wiechert *wiechert@simtec.mb.uni-siegen.de*
Edit. Board SNE	Thorsten Pawletta, *pawel@mb.hs-wismar.de*
Web EUROSIM	Anna Mathe, *anna.mathe@tuwien.ac.at*

Last data update March 2009

ASIM Working Groups. ASIM, part of GI - Gesellschaft für Informatik, is organised in Working Groups, dealing with applications and comprehensive subjects:

ASIM Working Groups

GMMS	Methods in Modelling and Simulation Peter Schwarz, *schwarz@eas.iis.fhg.de*
SUG	Simulation in Environmental Systems Wittmann, *wittmann@informatik.uni-hamburg.de*
STS	Simulation of Technical Systems H.T.Mammen, *Heinz-Theo.Mammen@hella.com*
SPL	Simulation in Production and Logistics Sigrid Wenzel, *s.wenzel@uni-kassel.de*
SVS	Simulation of Transport Systems U. Brannolte, *Brannolte@bauing.uni-weimar.de*
SBW	Simulation in OR C. Böhnlein, *boehnlein@wiinf.uni-wuerzburg.de*
EDU	Simulation in Education/Education in Simulation W. Wiechert, *wiechert@simtec.mb.uni-siegen.de*

CROSSIM – Croatian Society for Simulation Modelling

CROSSIM-*Croatian Society for Simulation Modelling* was founded in 1992 as a non-profit society with the goal to promote knowledge and use of simulation methods and techniques and development of education. CROSSIM is a full member of EUROSIM since 1997.

→ www.eurosim.info

vdusak@foi.hr

CROSSIM / Vesna Dušak
Faculty of Organization and
Informatics Varaždin, University of Zagreb
Pavlinska 2, HR-42000 Varaždin, Croatia

CROSSIM Officers

President	Vesna Dušak, *vdusak@foi.hr*
Vice president	Jadranka Božikov, *jbozikov@snz.hr*
Secretary	Vesna Bosilj-Vukšić, *vbosilj@efzg.hr*
Executive board members	Vlatko Čerić, *vceric@efzg.hr*
	Tarzan Legović, *legovic@irb.hr*
Repr. EUROSIM	Vesna Dušak, *vdusak@foi.hr*
Edit. Board SNE	Vesna Dušak, *vdusak@foi.hr*
Web EUROSIM	Jadranka Bozikov, *jbozikov@snz.hr*

Last data update March 2009

CSSS – Czech and Slovak Simulation Society

CSSS -The *Czech and Slovak Simulation Society* has about 150 members working in Czech and Slovak national scientific and technical societies (*Czech Society for Applied Cybernetics and Informatics*, *Slovak Society for Applied Cybernetics and Informatics*). The main objectives of the society are: development of education and training in the field of modelling and simulation, organising professional workshops and conferences, disseminating information about modelling and simulation activities in Europe. Since 1992, CSSS is full member of EUROSIM.

→ `www.fit.vutbr.cz/CSSS`

snorek@fel.cvut.cz

⌧ CSSS / Miroslav Šnorek, CTU Prague
FEE, Dept. Computer Science and Engineering,
Karlovo nam. 13, 121 35 Praha 2, Czech Republic

CSSS Officers

President	Miroslav Šnorek, *snorek@fel.cvut.cz*
Vice president	Mikuláš Alexík, *alexik@frtk.fri.utc.sk*
Treasurer	Evžen Kindler, *ekindler@centrum.cz*
Scientific Secr.	A. Kavička, *Antonin.Kavicka@upce.cz*
Repr. EUROSIM	Miroslav Šnorek, *snorek@fel.cvut.cz*
Deputy	Mikuláš Alexík, *alexik@frtk.fri.utc.sk*
Edit. Board SNE	Mikuláš Alexík, *alexik@frtk.fri.utc.sk*
Web EUROSIM	Petr Peringer, *peringer@fit.vutbr.cz*

Last data update December 2008

FRANCOSIM – Société Francophone de Simulation

FRANCOSIM was founded in 1991 and aims to the promotion of simulation and research, in industry and academic fields. Francosim operates two poles.

- Pole Modelling and simulation of discrete event systems. Pole Contact: *Henri Pierreval, pierreva@imfa.fr*
- Pole Modelling and simulation of continuous systems. Pole Contact: *Yskandar Hamam, y.hamam@esiee.fr*

→ `www.eurosim.info`

`y.hamam@esiee.fr`

⌧ FRANCOSIM / Yskandar Hamam
Groupe ESIEE, Cité Descartes,
BP 99, 2 Bd. Blaise Pascal,
93162 Noisy le Grand CEDEX, France

FRANCOSIM Officers

President	Yskandar Hamam, *y.hamam@esiee.fr*
Treasurer	François Rocaries, *f.rocaries@esiee.fr*
Repr. EUROSIM	Yskandar Hamam, *y.hamam@esiee.fr*
Edit. Board SNE	Yskandar Hamam, *y.hamam@esiee.fr*

Last data update April 2006

DBSS – Dutch Benelux Simulation Society

The Dutch Benelux Simulation Society (DBSS) was founded in July 1986 in order to create an organisation of simulation professionals within the Dutch language area. DBSS has actively promoted creation of similar organisations in other language areas. DBSS is a member of EUROSIM and works in close cooperation with its members and with affiliated societies.

→ `www.eurosim.info`

`a.w.heemink@its.tudelft.nl`

⌧ DBSS / A. W. Heemink
Delft University of Technology, ITS - twi,
Mekelweg 4, 2628 CD Delft, The Netherlands

DBSS Officers

President	A. Heemink, *a.w.heemink@its.tudelft.nl*
Vice president	W. Smit, *smitnet@wxs.nl*
Treasurer	W. Smit, *smitnet@wxs.nl*
Secretary	W. Smit, *smitnet@wxs.nl*
Repr. EUROSIM	A. Heemink, *a.w.heemink@its.tudelft.nl*
Deputy	W. Smit, *smitnet@wxs.nl*
Edit. Board SNE	A. Heemink, *a.w.heemink@its.tudelft.nl*

Last data update April 2006

HSS – Hungarian Simulation Society

The Hungarian Member Society of EUROSIM was established in 1981 as an association promoting the exchange of information within the community of people involved in research, development, application and education of simulation in Hungary and also contributing to the enhancement of exchanging information between the Hungarian simulation community and the simulation communities abroad. HSS deals with the organization of lectures, exhibitions, demonstrations, and conferences.

→ `www.eurosim.info`

`javor@eik.bme.hu`

⌧ HSS / András Jávor,
Budapest Univ. of Technology and Economics,
Sztoczek u. 4, 1111 Budapest, Hungary

HSS Officers

President	András Jávor, *javor@eik.bme.hu*
Vice president	Gábor Szűcs, *szucs@itm.bme.hu*
Secretary	Ágnes Vigh, *vigh@itm.bme.hu*
Repr. EUROSIM	András Jávor, *javor@eik.bme.hu*
Deputy	Gábor Szűcs, *szucs@itm.bme.hu*
Edit. Board SNE	András Jávor, *javor@eik.bme.hu*
Web EUROSIM	Gábor Szűcs, *szucs@itm.bme.hu*

Last data update March 2008

PSCS – Polish Society for Computer Simulation - update

PSCS was founded in 1993 in Warsaw. PSCS is a scientific, non-profit association of members from universities, research institutes and industry in Poland with common interests in variety of methods of computer simulations and its applications. At present PSCS counts 257 members.

→ `www.ptsk.man.bialystok.pl`

☞ `leon@ibib.waw.pl`

✉ PSCS / Leon Bobrowski, c/o IBIB PAN,
ul. Trojdena 4 (p.416), 02-109 Warszawa, Poland

PSCS Officers

President	Leon Bobrowski, *leon@ibib.waw.pl*
Vice president	Andrzej Grzyb, Tadeusz Nowicki
Treasurer	Z. Sosnowski, *zenon@ii.pb.bialystok.pl*
Secretary	Zdzislaw Galkowski, *Zdzislaw.Galkowski@simr.pw.edu.pl*
Repr. EUROSIM	Leon Bobrowski, *leon@ibib.waw.pl*
Deputy	A.Chudzikiewicz, *ach@it.pw.edu.pl*
Edit. Board SNE	Z.Sosnowski, *zenon@ii.pb.bialystok.pl*
PSCS Board Members	R. Bogacz , Z. Strzyzakowski Andrzej Tylikowski

Last data update March 2009

ISCS – Italian Society for Computer Simulation

The Italian Society for Computer Simulation (ISCS) is a scientific non-profit association of members from industry, university, education and several public and research institutions with common interest in all fields of computer simulation.

→ `www.eurosim.info`

☞ `Mario.savastano@uniina.at`

✉ ISCS / Mario Savastano,
c/o CNR - IRSIP,
Via Claudio 21, 80125 Napoli, Italy

ISCS Officers

President	MarioSavastano, *mario.savastano@unina.it*
Vice president	F. Maceri, *Franco.Maceri@uniroma2.it*
Repr. EUROSIM	F. Maceri, *Franco.Maceri@uniroma2.it*
Edit. Board SNE	Mario Savastano, *mario.savastano@unina.it*

Last data update April 2005

SIMS – Scandinavian Simulation Society

SIMS is the *Scandinavian Simulation Society* with members from the four Nordic countries Denmark, Finland, Norway and Sweden. The SIMS history goes back to 1959. SIMS practical matters are taken care of by the SIMS board consisting of two representatives from each Nordic country. Iceland will be represented by one board member.

SIMS Structure. SIMS is organised as federation of regional societies. There are FinSim (Finnish Simulation Forum), DKSIM (Dansk Simuleringsforening) and NFA (Norsk Forening for Automatisering).

→ `www.scansims.org`

☞ `petfr@ida.liu.se`

✉ SIMS/Peter Fritzson, IDA, Linköping University,
58183, Linköping, Sweden

SIMS Officers

President	Peter Fritzson, *petfr@ida.liu.se*
Treasurer	Vadim Engelson, *vaden@ida.liu.se*
Repr. EUROSIM	Peter Fritzson, *petfr@ida.liu.se*
Edit. Board SNE	Esko Juuso, *esko.juuso@oulu.fi*
Web EUROSIM	Vadim Engelson, *vaden@ida.liu.se*

Last data update December 2008

SLOSIM – Slovenian Society for Simulation and Modelling

SLOSIM - Slovenian Society for Simulation and Modelling was established in 1994 and became the full member of EUROSIM in 1996. Currently it has 69 members from both slovenian universities, institutes, and industry. It promotes modelling and simulation approaches to problem solving in industrial as well as in academic environments by establishing communication and cooperation among corresponding teams.

→ `msc.fe.uni-lj.si/SLOSIM`

☞ `slosim@fe.uni-lj.si`

✉ SLOSIM / Rihard Karba, Faculty of Electrical Engineering, University of Ljubljana,
Tržaška 25, 1000 Ljubljana, Slovenia

SLOSIM Officers

President	Rihard Karba, *rihard.karba@fe.uni-lj.si*
Vice president	Leon Žlajpah, *leon.zlajpah@ijs.si*
Secretary	Aleš Belič, *ales.belic@fe.uni-lj.si*
Treasurer	Milan Simčič, *milan.simcic@fe.uni-lj.si*
Repr. EUROSIM	Rihard Karba, *rihard.karba@fe.uni-lj.si*
Deputy	B. Zupančič, *borut.zupancic@fe.uni-lj.si*
Edit. Board SNE	Rihard Karba, *rihard.karba@fe.uni-lj.si*
Web EUROSIM	Aleš Belič, *ales.belic@fe.uni-lj.si*

Last data update March 2009

UKSIM – United Kingdom Simulation Society

UKSIM has more than 100 members throughout the UK from universities and industry. It is active in all areas of simulation and it holds a biennial conference as well as regular meetings and workshops.

→ `www.uksim.org.uk`

`david.al-dabass@ntu.ac.uk`

⌧ UKSIM / Prof. David Al-Dabass
Computing & Informatics,
Nottingham Trent University
Clifton lane, Nottingham, NG11 8NS
United Kingdom

UKSIM Officers

President	David Al-Dabass, *david.al-dabass@ntu.ac.uk*
Secretary	A. Orsoni, *A.Orsoni@kingston.ac.uk*
Treasurer	B. Thompson, *barry@bjtcon.ndo.co.uk*
Membership chair	K. Al-Begain, *kbegain@glam.ac.uk*
Univ. liaison chair	R. Cheng, *rchc@maths.soton.ac.uk*
Repr. EUROSIM	Richard Zobel, *r.zobel@ntlworld.com*
Edit. Board SNE	Richard Zobel, *r.zobel@ntlworld.com*

Last data update March 2009 (partially)

CEA-SMSG – Spanish Modelling and Simulation Group

CEA is the Spanish Society on Automation and Control In order to improve the efficiency and to deep into the different fields of automation, the association is divided into thematic groups, one of them is named 'Modelling and Simulation', constituting the group.

→ `www.cea-ifac.es/wwwgrupos/simulacion`

`simulacion@cea-ifac.es`

⌧ CEA-SMSG / María Jesús de la Fuente,
System Engineering and AutomaticControl department,
University of Valladolid,
Real de Burgos s/n., 47011 Valladolid, SPAIN

CAE - SMSG Officers

President	María J. la Fuente, *maria@autom.uva.es*
Repr. EUROSIM	Emilio Jiminez, *emilio.jiminez@unirioja.es*
Edit. Board SNE	Emilio Jiminez, *emilio.jiminez@unirioja.es*

Last data update March 2009

LSS – Latvian Simulation Society

The Latvian Simulation Society (LSS) has been founded in 1990 as the first professional simulation organisation in the field of Modelling and simulation in the post-Soviet area. Its members represent the main simulation centres in Latvia, including both academic and industrial sectors.

→ `briedis.itl.rtu.lv/imb/`

`merkur@itl.rtu.lv`

⌧ LSS / Yuri Merkuryev, Dept. of Modelling and Simulation Riga Technical University
Kalku street 1, Riga, LV-1658, LATVIA

LSS Officers

President	Yuri Merkuryev, *merkur@itl.rtu.lv*
Repr. EUROSIM	Yuri Merkuryev, *merkur@itl.rtu.lv*
Edit. Board SNE	Yuri Merkuryev, *merkur@itl.rtu.lv*

Last data update December 2008

ROMSIM – Romanian Modelling and Simulation Society

ROMSIM has been founded in 1990 as a non-profit society, devoted to both theoretical and applied aspects of modelling and simulation of systems. ROMSIM currently has about 100 members from both Romania and Republic of Moldavia.

→ `www.ici.ro/romsim/`

`sflorin@ici.ro`

⌧ ROMSIM / Florin Stanciulescu,
National Institute for Research in Informatics, Averescu Av. 8 – 10, 71316 Bucharest, Romania

ROMSIM Officers

President	Florin Stanciulescu, *sflorin@ici.ro*
Vice president	Florin Hartescu, flory@ici.ro Marius Radulescu, *mradulescu@ici.ro*
Secretary	Zoe Radulescu, *radulescu@ici.ro*
Repr. EUROSIM	Florin Stanciulescu, *sflorin@ici.ro*
Deputy	Florin Hartescu, *flory@ici.ro*
Edit. Board SNE	Florin Stanciulescu, *sflorin@ici.ro*

Last data update March 2009

ASIM − German Simulation Society

ASIM (*Arbeitsgemeinschaft Simulation*) is the association for simulation in the German speaking area, servicing mainly Germany, Switzerland and Austria. ASIM was founded in 1981 and has now about 700 individual members, and 40 institutional or industrial members (plus about about 300 affiliated members). The electronic *ASIM Newsletter* (three times a year) can be downloaded freely from ASIM website. For details/contacts, see Societies Short Info before.

SNE − Simulation News Europe. ASIM is publishing together with EUROSIM and ARGESIM the journal SNE, which is regularly sent to all ASIM members (as part of their membership; 900 issues) and spread for promotion purposes at conferences (300 issues). The electronic version *eSNE* is available to all members in high-resolution quality.

ASIM Books/ASIM Notes. ASIM co-operates with Springer Verlag (Berlin), Shaker Verlag (Aachen), and with ARGESIM Publishers (Vienna University of Technology) in publication of book series (Fortschritte in der Simulationstechnik - Frontiers in Simulation and Fortschrittsberichte Simulation - Advances in Simulation) and in publication of Proceedings. The trademark *ASim Mitteilungen* (ASIM Notes) stands for all publications of ASIM and ASIM Working Groups, in order to mark publications within the 'ASIM environment'. At present (2008 – 2010), an extended review is undertaken in order to classify all publications since 1982, and to make them electronically available for ASIM members, in full form or abstract form (details will be given in next SNE issue).

ASIM Working Groups. ASIM is part of GI − Gesellschaft für Informatik (Society for Informatics) and is itself structured into working groups (WG), which address various areas of modelling and simulation (attached conference/workshop reports).

ASIM Conferences. ASIM organises the conference series *Symposium Simulation Technique* (also known as previously annual ASIM Conference), the ASIM working groups organise annual workshops (up to 150 participants) and bi-annual conferences (more than 150 participants. ASIM cooperates in organising the tri-annual EUROSIM Congress and other EUROSIM and SCS conferences. Furthermore, ASIM co-organises special conferences, e.g. the bi-annual ASIM Wismar Workshop, and the the three-annual conference series MATHMOD − Mathematical Modelling in Vienna (report MATHMOD 2009 next SNE issue).

Upcoming Conferences. At present, the following conferences and workshops with ASIM as organizer, co-organizer or co-sponsor are scheduled for 2009/2010:

- ASIM 20th *Symposium Simulation Technique*,
  September 23 -25, 2009, Cottbus, Germany
- 2nd ASIM SPL Workshop *Planning and Simulation in Logitic Application*, at GI Annual Conference, Oct.1, 2009; Lübeck, Germany
- 13th ASIM SUG Workshop *Modelling and Simulation of Ecosystems*,
  October 28 - 30, 2009, Usedom, Germany
- ASIM GMMS Workshop *Computational Science and Engineering*,
  March 3 – 5, 2010, Jülich, Germany
- ASIM STS Workshop *Simulation of Technical Systems*, March 4 – 5, 2010, Ulm, Germany
- EUROSIM 2010 − *7th EUROSIM Congress*
  Sept. 5 – 9, 2010, Prague, Czech Republic
- 14th ASIM *SPL Conference Simulation in Production and Logistics*,
  October 7 – 8, 2010, Karlsruhe, Germany

ASIM 2009 CONFERENCE
20th Symposium Simulation Technique
September 23- 25, 2009, Cottbus, Germany
`www.asim-gi.org`, `www.tu-cottbus.de/asim2009`

ASIM 2009, the 20th ASIM *Symposium Simulation Technique*, will be organised by Albrecht Gnauck at Brandenburgische Technische Universität Cottbus, September 23 – 25, 2009. The conference themed *Simulation for Environment, Climate, Energy and Techniques* will provide plenary lectures, parallel sections, workshops, poster exhibition, tool exhibitions, and excursions. Furthermore, an ASIM *General Assembly* is scheduled.

Up to now, also special sessions to Environmental Modelling, E-Learning with/for Simulation, Trends in Logistic Simulation, and Physical Modelling are planned. Conference language is German and English, contributions will be published in peer-reviewed Proceedings. Participants are invited to excursions to Spreewald, Surface Mining, Power Station, and to other social events. For details, deadlines, and submission see the conference website `www.tu-cottbus.de/asim2009`.

CONFERENCE REPORT

13th ASIM Conference on Simulation in Production and Logistics, Berlin, October 2008

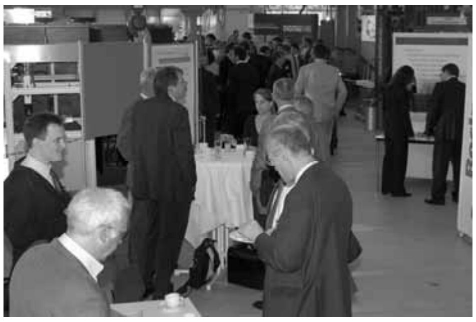
On October 1st and 2nd, 2008 the ASIM conference dedicated to the production and logistics applications of discrete-event simulation (ASim-SPL) took place in Berlin (Germany). The conference, now in Berlin for the fifth time, was organized by Dr. Markus Rabe from Fraunhofer IPK and attracted around 200 researchers and practitioners mainly from Germany and Austria, but also from Switzerland, The Netherlands, Belgium, Ireland, Finland, Spain, Serbia, Ukraine, and USA.

Johann Bayer
BMW AG München

In the first plenary speech, Johann Bayer (BMW AG) gave insight into BMW's new sales and production system. He highlighted the current challenges, like growing customer demand towards more individuality and service, the broadening of the product portfolio, shorter model cycles, new forms of sales like agents and e-commerce, and many others. On the vision that any customer should receive an individually ordered car at a binding date, Johann Bayer discussed the implications of this strategy on production and its validation within the production through simulation techniques.

In the evening of the first day, the participants enjoyed a great speech from Prof. Manfred Spitzer (Ulm), illustrating how our experience does modify our brain, physically. The implications for learning in kindergarten, school, university and business have been discussed, and the relationship studied between fear and learning as well as joy and learning. With fascinating examples, Manfred Spitzer demonstrated how our brain can process complex information and derive the essential (and survival-relevant) consequences.

The second plenary was a guest speech from the ASIM working group on technical systems simulation, presented by Prof. Walter Commerell (Ulm). Techniques for modelling and simulation of technical systems have gained steadily increasing importance in the recent years, targeting to develop innovative and optimised products under a high pressure of cost and time constraints. This is especially true for the design of mechatronic systems. In this field, simulation is established as an important part of the development process. The speech has illustrated the application potentials of simulation at the example of a mechatronic system from a passenger car. Solutions have been presented that enable domain-overspanning simulation. Furthermore, possible methods to achieve a non-interrupted, simulation-supported design process were presented. Current challenges in simulation of technical systems were shown and trends and perspectives derived for future developments.


Exhibition in Fraunhofer IPK's Shop Floor Area

The main part of the conference was formed by about 60 paper presentations in four parallel streams. The streams covered the major topics like:

- Digital factory
- New methods and techniques
- Emulation
- Optimisation
- Distributed simulation

Also, several streams were focussed on specific application areas:

- Automotive industry
- Manufacturing and workshop applications
- Logistics applications
- Supply chain applications
- Personnel simulation
- Shipbuilding and operation of vessels
- Construction industry

An exhibition with nine exhibitors accompanied the conference, which raised high interest and intense discussions.

As always with this conference, an evening event provided not only room for discussions and good food and drinks, but also entertainment; this time with a-capella music presented by the Berlin group "Männerwirtschaft" as well as by magicians.

Evening Event, Universal Hall Berlin

Pre-conference day, tutorial presentation: Dr. Markus Rabe

For the first time, a "pre-conference day" (September 30) was organized, offering advanced tutorials. In the first tutorial, Prof. Oliver Rose from the Technical University Dresden presented hints and examples for statistics in simulation studies. After a coffee break, Dr. Markus Rabe introduced methods and techniques for verification and validation. Both tutorials have been fully booked, demonstrating the high interest in this type of conference enrichment. This pre-conference day closed with a get-together reception.

The conference book is still available through book stores or from Fraunhofer IPK at the price of € 78,00. Members of ASIM may order at a special price of € 68,00 which is only available upon direct order at Fraunhofer IPK (tagung@asim.fraunhofer.de). At the conference website www.asim.fraunhofer.de, additional information can be found about past ASIM conferences and related publications.

The next ASIM conference on *Simulation in Production and Logistics* (SPL) is planned for autumn 2010, with the conference venue to be decided in spring 2009. For 2012, the *Winter Simulation Conference* is planned for Berlin (Germany), leading to a shift of the the 15th ASIM SPL conference to 2013.

CONFERENCE REPORT

4th ASIM Wismar Workshop, May 2008

The ASIM working group *Simulation of Technical Systems (STS)* and Mothods in Modelling and Simulation held the 4th ASIM Workshop *Modellierung, Regelung und Simulation in Automotive und Prozessautomation* in Wismar on May 29–30, 2008. About 140 engineers and scientists from industry, research labs and university met for a common meeting at Hochschule Wismar.

Two plenary papers introduced into the main topics:

- *Use the Simulation at the example of the hybrid-motor-development* by W. Nietschke, IAV GmbH Gifhorn
- *Possibilities and constraints of the use of Modelica at the example of thermal systems* by Dr. W. Tegethoff, TLK-Thermo GmbH Braunschweig

Furthermore two tutorials were held about the topics:

- Modelling approach to the system simulation
- An introduction in Modelica/Dymola for beginners

About 40 presentations introduced to intense discussions of the following subjects:

- Language standards, e.g.VHDL-AMS and Modelica
- Modeling and simulation in Automotive Systems

Parts of the exhibition area

- Tools and applications
- Special applications and thermal systems
- Mechatronic systems
- Medical technology
- Statistical analysis
- Model based development of functions

Six exhibitors showed their software and their services.

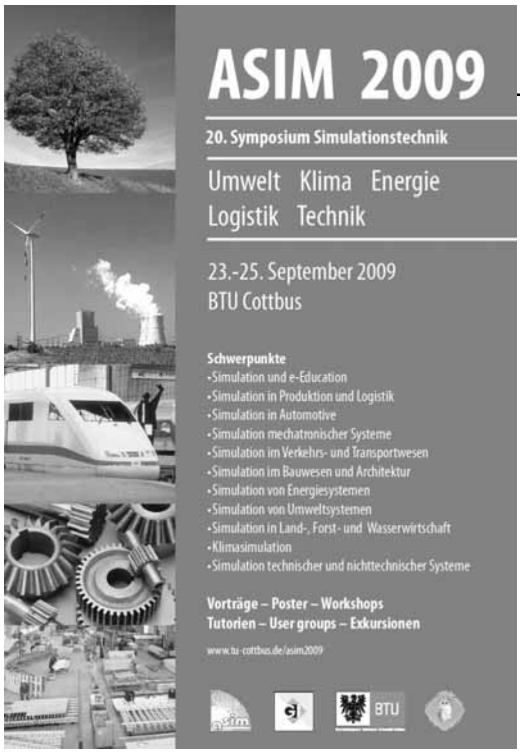
The evening of the first day was spent with a traditional meal in the restaurant “Brauhaus”. The meeting was closed with two excursions –Historic city guided tour to the hanseatic league and brick-gothic and a drive with the Poeler Kogge Wissemara.

The participants had much time for discussion and the chance to make new contacts and also to meet old friends. The meeting was organised by the Research Group Computational Engineering & Automation of Hochschule Wismar, the IAV GmbH Gifhorn and by ASIM.

In 2009, the working group will organise the annual workshop at University of Dresden (organiser Prof. Oliver Rose), together with Fraunhofer IIS/EAS Dresden (organiser Dr. Joachim Haase).

Look on the historical city

Work on the historical Poeler Kogge Wissemara


ASIM 2009

20. Symposium Simulationstechnik

Umwelt Klima Energie
Logistik Technik

23.-25. September 2009
BTU Cottbus

Schwerpunkte
- Simulation und e-Education
- Simulation in Produktion und Logistik
- Simulation in Automotive
- Simulation mechatronischer Systeme
- Simulation im Verkehrs- und Transportwesen
- Simulation im Bauwesen und Architektur
- Simulation von Energiesystemen
- Simulation von Umweltsystemen
- Simulation in Land-, Forst- und Wasserwirtschaft
- Klimasimulation
- Simulation technischer und nichttechnischer Systeme

Vorträge – Poster – Workshops
Tutorien – User groups – Exkursionen

www.tu-cottbus.de/asim2009

Call for ASIM 2009 Conference

20th Symposium Simulation Technique

September 23 – 25, 2009, Cottbus, Germany

www.asim-gi.org, www.tu-cottbus.de/asim2009

SLOSIM – Slovenian Society for Simulation and Modelling

SLOSIM, the *Slovenian Society for Simulation and Modelling*, was established in 1994 and became the full member of EUROSIM in 1996. Currently, it has 76 members from Slovenian universities, institutes, and industry. It promotes modelling and simulation approach to problem solving in industrial as well as in academic environments by establishing communication and cooperation among the corresponding teams. Organisational details and contact addresses can be found in the SLOSIM short information (see before).

→ `msc.fe.uni-lj.si/slosim`

`slosim@fe.uni-lj.si`

Activities

As planned, the excursion to the Ljubljana international airport was organized for the SLOSIM members on March, 28th. Twenty participants had the opportunity to visit the flight simulator for three different types of aircrafts and presentation of virtual air-space of Slovenia and Europe. See the virtual airspace of Slovenia and Europe and attend the lecture organized by The International Virtual Aviation Organization, Slovenian division. (http://www.ivao.si/en/index.php).

Past Events

In the last period SLOSIM actively participated in the organization of the 6th *Vienna Int. Conference on Mathematical Modelling* – MATHMOD 09. R. Karba as the IPC member, some colleagues as the reviewers and G. Mušič as the organizer of the tutorial: DES and Petri Nets in MATLAB. A. Belič had a plenary lecture: *Modelling in Biology, Neurology and Pharmacy*. In sum, SLOSIM members participated with 13 contributions. As the society organized the special sessions on all MATHMOD events in the last one M. Atanasijević–Kunc and G. Mušič were organizers of the following two track sessions:

- Modelling, Simulation and System Dynamics through E-Learning (11 papers),
- Discrete and Hybrid Simulation Methodology, techniques and Applications (11 papers).

Coming Events

Annual ERK (Electrotechnical and Computer Science Conference) will be organised at the end of September 2009 in Portorož, Slovenia. SLOSIM will participate with several sessions.

SimS − Scandinavian Simulation Society

SIMS is the Scandinavian Simulation Society with members from the four Nordic countries Denmark, Finland, Norway and Sweden. The SIMS history goes back to 1959. SIMS practical matters are taken care of by the SIMS board consisting of two representatives from each Nordic country. Iceland will be represented by one board member. Organizational details and contact addresses see short information before.

→ `www.scansims.org`

`esko.juuso@oulu.fi`

SIMS Structure. SIMS is organised as federation of regional societies. There are FINSIM (Finnish Simulation Forum), DKSIM (Dansk Simuleringsforening) and NFA (Norsk Forening for Automatisering).

Past Events

In spring 2009 some workshops and special conferences took place (reports in the next SNE issue):

- **NPCW '09 -** 15th Nordic Process Control Workshop, Jan. 29-30, 2009. Porsgrunn, Norway
- OpenModelica Annual Workshop, February 2, 2009, Linköping, Sweden
- ModProd 2009 - 3rd Annual Workshop on Model-based Product Development, February 3-4, 2009, Linköping, Sweden;
- Automation XVIII Seminar, March 17-18, 2009, Helsinki, Finland

Coming Events

The 50th Scandinavian Conference on Simulation and Modelling, will be organized by DKSIM in Copenhagen, Denmark, October 6-8, 2008. The purpose of the SIMS conference is to cover broad aspects of modeling and simulation and scientific computation. It will be of interest for model builders, simulator personnel, scientists, engineers, vendors, etc.

SIMS 2009

50th Conference on Simulation and Modelling

October 6-8, 2009, Copenhagen, Denmark

`www.scansims.org`

The scientific program will consist of technical sessions (submitted and invited papers), and is open for poster sessions and vendor demonstrations. The focus area is modelling and simulation in energy systems.

Proceedings of the accepted papers will be distributed at the conference.

Presented papers will be considered for publication in the EUROSIM scientific journal '*Simulation and Modelling – Practice and Theory*' (SIMPRA) published by Elsevier Science. Especially Ph.D. students are encouraged to contribute with papers according to the conference themes.

CEA-SMSG − Spanish Modeling and Simulation Group

IFAC is the *International Federation of Automatic Control*, a multinational federation of organizations representing the engineering and scientific societies concerned with automatic control.

CEA is the *Spanish Society on Automation and Control* and it is the national member of IFAC in Spain. Since 1968 CEA-IFAC looks after the development of the Automation in Spain, in its different issues: automatic control, robotics, simulation, etc. In order to improve the efficiency and to deep into the different fields of Automation, the association is divided into thematic groups, concretely eight groups at present.

One of them is named *Modelling and Simulation*, constituting then the CEA-SMSG (CEA-IFAC *Spanish Modelling and Simulation Group*), which looks after the development of modelling and simulation in Spain. This group works basically about all the issues concerning the use of modelling and simulation techniques as essential engineering tools for decision-making.

The coordinator of the group is Dra. María Jesús de la Fuente, from the University of Valladolid. The representative of the group in EUROSIM is Dr. Emilio Jiménez Macías, from the University of La Rioja: `emilio.jimenez@unirioja.es`

`simulacion@cea-ifac.es`

Activities

The main usual activities of the group can be summarized as an annual meeting about modelling and simulation, inside CEA meeting on automation, specialized courses, a distribution list, a periodic electronic report, technical books, a journal (translated as *Latin American journal of Automation and Industrial Computing*), a trade agreement with Pearson Inc. for a collection of books, an award for the scientific contribution in automation and a specific award for modelling and simulation, sponsorship of events, etc.

Past Events

As most interesting recent activities could be selected the 29th annual meeting, inside the National Automatic Workshop (XXIX Jornadas de Automática) by the CEA-IFAC. September 2008, Tarragona (Spain), which includes the annual meeting of the CEA-SMSG (CEA-IFAC Spanish Modeling and Simulation Group).

Coming Events

30th annual meeting is scheduled inside the National Automatic Workshop (XXX Jornadas de Automática) by the CEA-IFAC, in Valladolid, September 2009, which includes the annual meeting of the CEA-SMSG.

The 6th International Mediterranean Modelling Multiconference, I3M 2009 (EMMS 2009, MAS 2009, HMS 2009) takes place in Puerto de la Cruz (Tenerife, Islas Canarias) September 23–25, 2009. It includes:

- The 21st European Modelling & Simulation Symposium - EMSS 2009
  `i3m2009.isaatc.ull.es/emss2009`
- The International Workshop on Modeling & Applied Simulation- MAS 2009
  `http://i3m2009.isaatc.ull.es/mas2009`

ROMSIM − Romanian Society for Modelling and Simulation

ROMSIM has been founded in 1990 as a non-profit society, devoted to both theoretical and applied aspects of modelling and simulation of systems. ROMSIM currently has about 100 members from both Romania and Republic of Moldavia.

The main objectives of ROMSIM are: development of new methods and instruments of modelling and simulation of systems, development of new application of modelling and simulation of both natural systems and those created by man, development of education and training in the field of modelling and simulation of systems.

In April 1999 ROMSIM has been accepted as an observer member of EUROSIM.

→ `http://www.ici.ro/romsim/`

`sflorin@ici.ro`

Activities

ROMSIM is involved in organization of two periodic scientific seminaries: A seminary on Systems modeling and Simulation and a seminary on Mathematical modeling Simulation; the seminaries are attended each time by 15 to 20 specialists. During the seminaries participants present and discuss both theoretical and applied contributions in the field of systems modeling and simulation.

An important objective of ROMSIM is organization of national scientific events in the field of modelling and simulation and participation at international conferences.

Past Events

In 2008 ROMSIM was involved in the organization of two international conferences:

- International Conference of Differential Geometry and Dynamical Systems (DGDS-2008)
- 5th International Colloquium of Mathematics in Engineering and Numerical Physics (MENP-5) August 29 – Sept. 02, 2008 at Callatis High School, Mangalia, near the Black Sea, Romania

Marius Radulescu was a member of the Scientific and Organizing Committee of the above conferences. Conference Program Committee includes members from France, Greece, India, Italy, Japan, Macedonia-FYROM, Portugal, Russia, Turkey, Spain, Yugoslavia, U.K. and USA.

Members of ROMSIM gave talks to several international conferences in 2008 with subjects in the domain of mathematical modelling and simulation - Tenth International Conference on Computer Modeling and Simulation (UKSIM 2008) Cambridge UK (one paper), - 9th WSEAS International Conference on Mathematics and Computers in Business and Economics (MCBE'2008), Bucharest, (three papers), - 12th WSEAS International Conference on Computers, Heraklion, Greece, 2008 (three papers), - Operations Research, OR2008 conference in Augsburg, Germany, - and MIA 2008 conference in Trogir, Croatia

Coming Events

DGDS-2009

International Conference of Differential Geometry and Dynamical Systems

October 8-11, 2009, Bucharest, Romania

`www.mathem.pub.ro/dept/dgds-09/conf.htm`

Publications

We emphasize also the activity of ROMSIM members in the field of publishing articles in international and/or Romanian journals. In 2008, 6 papers were published in *WSEAS Proceedings* (ISI indexed), one paper in Proc. Romanian Academy (ISI indexed) and one paper in the Journal Studies in Informatics and Control. At the same time some members of ROMSIM (as dr. Florin Hartescu, dr. Constantza Zoie Radulescu, et al) have published research articles in *Romanian Journal for Informatics and Automatics*.

Several editorial events must be emphasized. One is the publication of the book entitled *Modeling of High Complexity Systems with Applications* (including a CD with 12 MathCAD applications) by WIT Press, Southampton, Boston, authored by Dr. Florin Stanciulescu.

Another is the publication by the Romanian Academy Publishing House of the book Mathematical Models for Optimal Asset Allocation; Authors are Marius Radulescu, Sorin Radulescu and Constantza Zoie Radulescu. One book with a subject connected to ruin theory (which has some simulation models), author Gheorghita Zbaganu was published in a collection of the Publishing House Geometry Balkan Press.

CroSSim – Croatian Simulation Society

CroSSim – *Croatian Society for Simulation Modelling* was founded in 1992 as a non-profit society with the goal to promote knowledge and use of simulation methods and techniques and development of education and training in the field of simulation modelling. CroSSim is a full member of EuroSim since 1997.

→ www.eurosim.info

✉ vdusak@foi.hr

Publications. CroSSim co-operates with the University Computing Centre, Zagreb, in publishing of the Journal of Computing and Information Technology (CIT). All information concerning CIT is available at cit.srce.hr.

Past Events. 30th International Conference *Information Technology Interfaces ITI 2008* took place in Cavtat near Dubrovnik, 23-26 June 2008. It included four sessions on Modeling, Simulation and Optimization chaired by Felix Breitenecker, Želimir Kurtanjek and Robert Manger.

In addition an invited anniversary lecture was given by Professor Felix Breitenecker entitled *Love Emotions between Laura and Petrarca – an Approach by System Dynamics* and co-authored by Florian Judex, Nikolas Popper, Andreas and Anna Mathe.

Coming Events. 31st International Conference Information Technology Interfaces ITI 2009 will be held in Cavtat near Dubrovnik on 22-25 June 2009. Modeling, Simulation and Optimization is among topics of interest and a special session is devoted to Medical Informatics.

ITI 2009

31st International Conference

Information Technology Interfaces

June 25 – 27, 2009

Cavtat near Dubrovnik, Croatia

iti.srce.hr

PSCS – Polish Society for Computer Simulation

PSCS, the *Polish Society for Computer Simulation*, was founded in 1993 in Warsaw. PSCS is a scientific, non-profit association of members from universities, research institutes and industry in Poland with common interests in variety of methods of computer simulations and its applications. At present PSCS counts 257 members.

→ www.ptsk.man bialystok.pl

✉ PSCS/ Leon Bobrowski, c/o IBIB PAN,
ul. Trojdena 4 (p. 416), 02-109, Warszawa, Poland

Activities. The main activities of the Polish Society for Computer Simulation are annual conferences known as *PSCS Workshops on Simulation in Research and Development*. The PSCS Workshops were organized in: Mielno (1994), Warszawa(1995), Wigry (1996), Jelenia Gora (1997, 1998), Bialystok & Bialowieza (1999), Zakopane – Koscielisko (2000), Gdansk-Sobieszewo (2001), Osieki k/ Koszalina (2002), Zakopane (2003), Białystok & Augustow (2004), Sarbinowo Morskie k/Koszalina (2005), Krynica Zdroj (2006) and Zakopane (2008).

Past Events. On February 20, 2009 the general assembly of PSCS members was held in Warsaw. This meeting, besides representing an interesting forum to discuss and promote the activity of the society, was the occasion to elect the Board for the period 2009-2011.

Coming Events. Assoc. Prof. Walenty Oniszczuk will organize the 16th PSCS Workshop on *Simulation in Research and Development* on September 23-26, 2009 in Bialystok, Poland; ptsk2009@wi.pb.edu.pl

16TH PSCS Workshop

Simulation in Research and Development

September 23 – 26, 2009, Bialystok, Poland

www.ptsk.man.bialystok.pl

Publications. Proceedings of the 15th PSCS Workshop on *Simulation in Research and Development*, T. Nowicki and J. Koszela (Eds.), Warszawa, 2008, (in Polish). The price is 30,- PLN.

Conference Series – MathMod, EOOLT, Modelica, MoSim

MATHMOD Vienna Conference Series

At MathMod Vienna – *Vienna International Conference on Mathematical Modelling* scientists and engineers using or developing models or interested in the development or application of various modelling tools are offered an opportunity to present ideas, methods and results and discuss their experiences or problems with experts of various areas of specialisation. The scope of the *MathMod Conference Series* covers theoretic and applied aspects of various types of mathematical modelling. Comparison of modelling approaches, model simplification, modelling uncertainties, port-based modelling and the impact of items such as these on problem solution, numerical techniques, validation, automation of modelling and software support for modelling, co-simulation, etc are discussed.

MathMod 2009. The conference series *MathMod Vienna* - was started in February 1994 with the 1st MathMod Vienna − MathMod 1994, followed conferences in February 1997, 2000, 2003, and 2006. Recently MathMod 2009 took place: 451 participants from 44 countries followed 11 invited lectures, 291 session contributions, and 82 short papers / posters. A detailed report will be given in SNE 19/3-4, including the *MathMod & Arts* project. MathMod 2012 is scheduled for February 2012. www.mathmod.at

MOSIM – FRANCOSIM

Although FRANCOSIM, the French-speaking simulation society is undergoing a re-organization, the conference series MOSIM is still continued. MOSIM – *International Conference in Modelling and Simulation* – started in 1997 with 1st MOSIM in Rouen, followed by conferences in Annecy (1999), Troyes (2001), Toulouse (2003), Nantes (2004), Rabat (2006), and 7th MOSIM in Paris (2008). Each MOSIM conferences emphasizes on a general topic, e.g. MOSIM 2008 *with Communication, Cooperation, and Coordination*, or MOSIM 2006 on *Modelization, Optimization and Simulation of the Systems: Challenges and Opportunities*.

MoSim'10
8th Conference on Modelling and Simulation
Hammamet, Tunisia, May 10-12, 2010
www.enim.fr/mosim10

MOSIM'10. The *8th MOSIM* conference will be held in Hammamet in Tunisia, which is a perfect place for a fruitful international scientific event. This 2010 MOSIM edition is organized by LGIPM (Industrial Engineering and Production Laboratory of Metz, France) in collaboration with CEREP (Manufacturing Research Center, Tunisia). Conference theme is *Evaluation and Optimization of Innovative Production Systems of Goods and Services*. Proposition of special sessions and tracks is also encouraged – more information at website: www.enim.fr/mosim10.

Modelica Conference Series

Modelica is a freely available, object-oriented language for convenient and efficient modelling and simulation of complex, multi-domain physical systems, described by ordinary differential and algebraic equations. The freely available *Modelica Standard Library* provides a growing number of basic models covering all the fields of engineering. The development and promotion of Modelica and of the *Modelica Standard Library* is organized by the non-profit *Modelica Association*, bringing together tool developers, library developers and scientists from industry and academia since 1997.

The *Modelica Conference* series is intended to bring together people using Modelica for modelling, simulation, and control applications, Modelica language designers, Modelica tool vendors and Modelica library developers.

Modelica'2009
7th International Modelica Conference
Como, Italy, September 20 – 22, 2009
www.modelica.org/events/modelica2009

Modelica 2009. *Modelica Conferences* are organized with a rhythm of 18 month. *1st Modelica Conference* took place March 2002, Oberpfaffenhofen,, Germany, followed by Modelica conferences in Linköping, (Nov. 2003), Hamburg (March 2005), Vienna, (September 2006), and Bielefeld, Germany (March 2008).

The *Modelica Association* and *Politecnico di Milano* organize the *7th International Modelica Conference, Modelica'2009*, at Como, Italy, September 2009.

The conference will cover all the relevant Modelica topics: language design, numerical and symbolic methods, reusable model libraries, software tools, scientific and industrial applications. Tutorials will be held on Sunday afternoon, while regular sessions, poster sessions, tool presentations, and user's group meetings will be held on Monday and Tuesday. More info at `www.modelica.org/events/modelica2009`.

Equation-Based Object-Oriented Languages and Tools

Computer aided modeling and simulation of complex systems, using components from multiple application domains, such as electrical, mechanical, hydraulic, control, etc., have in recent years witnessed a significant growth of interest. In the last decade, novel equation-based modeling languages, (e.g. Modelica, gPROMS, Verilog-AMS, VHDL-AMS, and SysML) supporting acausal modeling using differential algebraic equations (DAEs) have appeared. In the last couple of years the name equation-based object-oriented (EOO) language has been introduced to denote modeling languages within this category.

The EOOLT Workshop Series addresses the current state of the art of EOO modeling languages as well as open issues that currently still limit the expression power, correctness, and usefulness of such languages through a set of full-length presentations and forum discussions.

EOOLT 2009
3rd Workshop *Equation-Based Object-Oriented Languages and Tools* – within MODELICA 2009
Como, Italy, September 23, 2009
`www.eoolt.org/2009`

EOOLT 2009. EOOLT Workshops are organized within conferences of similar, but more global subjects. After EOOLT 2007 (Berlin, July 2007) within ECOOP 2007 − *European Conference on Object-Oriented Programming*, and EOOLT 2008 within ECOOP 2008 (Paphos, Cyprus, July 2008), EOOLT 2009 will be organized within the MODELICA Conference 2009, Como, Italy, September 2009. More information see website: `www.eoolt.org`.
.

MISS and M&SNet – Initatives of SCS

MISS - the *McLeod Institute of Simulation Sciences* is an initiative of the *Society for Modeling and Simulation International* – SCS. The Institute is named after Mr. John McLeod, the founder of SCS. MISS consists of co-operating *MISS Centers* active in professionalism, research, education, and knowledge dissemination in the modeling and simulation domain.

The MISS aims to provide an organizational structure that will serve to integrate and enrich, within its Centers, the activities of modeling and simulation expertise throughout the world. More information from Andras Javor, MISS Director, javor@eik.bme.hu, and at web `www.scs.org`. Reports on *MISS Centers* and *M&SNet Nodes* will start in SNE 19/3-4.

M&SNet – *McLeod Modeling and Simulation Network* is a consortium of co-operating independent organizations active in modeling and simulation. It was established in 2003 by the *Society for Modeling and Simulation International*.

The M&SNet aims to provide an organizational structure that will serve to integrate and enrich, within its organizations, modeling and simulation activities throughout the world. The M&SNet provides a framework within which organizations interested in M&S can interact, share expertise, and work on problems of common interest. Info: Tuncer Ören, M&SNet Executive Director, `oren@site.uotawa.ca`, `www.scs.org/msnet`.

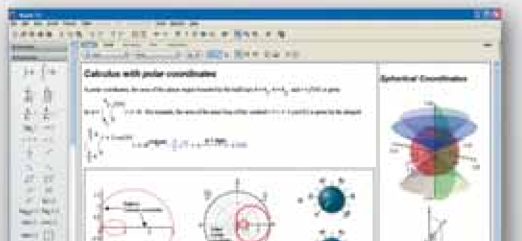
Both groups will meet officially on occasion of ISMc'09 on July 15, in order to discuss further activities. Working meetings are planned for I3M'2009.

ISMC'09
2009 International Simulation Multiconference
July 13 – 16, 2009, Istanbul, Turkey
w `www.scs.org/confernc/summersim/summersim09`

I3M2009
6th Mediterranean Modelling Multiconference
Sept. 23-25, 2009; Tenerife - Canary Islands, Spain
`i3m2009.isaatc.ull.es/emss2009`

Maple™ 12

The Essential Tool for Mathematics and Modeling

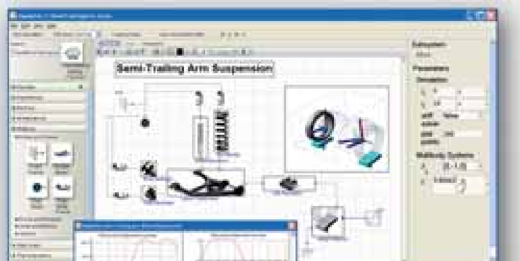


Ob Sie schnelle Lösungen für mathematische Probleme benötigen oder anspruchsvolle technische Dokumente und Applikationen erstellen möchten: Maple 12 bietet die Tools, um Ihre mathematischen Fragestellungen zu formulieren, zu lösen und Ihre Ergebnisse zu dokumentieren.

Die umfangreiche Auswahl an Zusatzprodukten zu Maple bietet Ihnen zudem die Möglichkeit, die Reichweite Ihrer Arbeit wesentlich auszubauen. Dazu gehört unter anderem die Maple Toolbox für MATLAB®, die Global Optimization Toolbox oder auch Maple T.A., ein Tool für Web-basiertes Lernen und Prüfen.

MapleSim™

High Perfomance Multi-Domain Modeling & Simulation



MapleSim ist eine fachübergreifende Multidomänen-Hochleistungssoftware für Modellierung und Simulation, die bei der Markteinführung neuer Produkte durchgreifende Erfolge erzielt. Die weltweit am höchsten entwickelte Software kombiniert symbolische mit numerischen Rechenmethoden und ermöglicht dadurch eine grundlegende Erneuerung im Simulations- und Modellierungsprozess.

scientific COMPUTERS

Scientific Computers GmbH • Friedlandstraße 18 • 52064 Aachen / Germany • Telefon (0241) 40008-0 • www.scientific.de • maple@scientific.de

515.000.000 KM, 380.000 SIMULATIONEN
UND KEIN EINZIGER TESTFLUG.
DAS IST MODEL-BASED DESIGN.

Nachdem der Endabstieg der beiden Mars Rover unter Tausenden von atmosphärischen Bedingungen simuliert wurde, entwickelte und testete das Ingenieur-Team ein ausfallsicheres Bremsraketen-System, um eine zuverlässige Landung zu garantieren. Das Resultat – zwei erfolgreiche autonome Landungen, die exakt gemäß der Simulation erfolgten.
Mehr hierzu erfahren Sie unter:
www. mathworks.de/mbd

MATLAB® & SIMULINK®

The MathWorks

Accelerating the pace of engineering and science