Enhancing Modelica towards variable structure systems

Dirk Zimmer

Institute of Computational Science, ETH Zürich,
CH-8092 Zürich, Switzerland
dzimmer@inf.ethz.ch

Abstract. This paper explains the motivation behind variable structure systems and analyses the current Modelica language with respect to those concerns. The major flaws and shortcomings are discussed to raise the awareness for the most relevant problem sets. Furthermore we sketch our current research activity in broad terms and explain our approach that consists of a new modeling language. Finally, a small example is presented.

Keywords: object-oriented modeling, variable structure systems.

1 Motivation

Many contemporary models contain structural changes at simulation run time. These systems are typically denoted by the collective term: variable structure systems. The motivations that lead to the generation of such systems are manifold:

- The structural change is caused by ideal switching processes. Classic examples are ideal switching processes in electric circuits, rigid mechanical elements that can break apart, e.g. a breaking pendulum or the reconfiguration of robot models [4].
- The model features a variable number of variables: This issue typically concerns social or traffic simulations that feature a variable number of agents or entities, respectively.
- The variability in structure is to be used for reasons of efficiency: A bent beam should be modeled in more detail at the point of the buckling and more sparsely in the remaining regions.
- The variability in structure results from user interaction: When the user is allowed to create or connect certain components at run time, this usually reflects a structural change.

The term variable structure system turns out to be a rather general term that applies to a number of different modeling paradigms, such as adaptive meshes in finite elements, discrete communication models of flexible computer networks, etc. We focus on the paradigm that is represented by Modelica: declarative models that are based on DAEs with hybrid extensions. Within such a paradigm, a structural change

is typically reflected by a change in the set of variables, and by a change in the set of relations (i.e., equations) between these time-dependent variables. These changes may lead to severe changes in the model structure. This concerns the causalization of the equation system, as well as the perturbation index of the DAE system.

A general modeling language supporting variable structure systems offers a number of important benefits. Such a potential language incorporates a general modeling methodology that enables the convenient capture of knowledge concerning variable structure systems, and provides means for organizing and sharing that knowledge both by industry and science. A corresponding simulator is a valuable tool for engineering and science education.

In concrete terms, our research is intended to aid the further extension of the Modelica framework. This benefits primarily the prevalent application areas of mechanics and electronics.

- Ideal switching processes in electronic circuits (resulting from ideal switches, diodes, and thyristors) can be more generally modeled. Occurring structural singularities can be handled at run time.
- The modeling of ideal transitions in mechanical models, like breaking processes or the transition from friction to stiction, become a more amenable task.

Additional applications may occur in domains that are currently foreign to Modelica. This might concern for instance:

- Hybrid economic or social simulations that contain a variable number of entities or agents, respectively.
- Traffic simulations.

Finally, more elaborate modeling techniques become feasible. For instance multi-level models can be developed, whereby the appropriate level of detail is chosen at simulation run time in response to computational demands and/or level of interest.

2 Analysis of Modelica

Unfortunately, the modeling of variable structure systems within the current Modelica framework is very limited. This is partly due to a number of technical restrictions that mostly originate from the static treatment of the DAEs. Specific techniques, like inline-integrations [2] can help in certain situations, but they do not provide a general solution. Although the technical restrictions represent a major limiting factor, other issues need to be concerned as well. An important problem is the lack of expressiveness in the Modelica-language.

To get a better understanding, we analyze the Modelica language with respect to the modeling of structural changes and list the most problematic points in the following subsections.

2.1 Lack of Conditional Declarations

Modelica is a declarative language that is based upon the declaration of equations, basic variables and sub-models. Modelica offers conditional blocks (i. e.: if, when) that enable the convenient formulation of changes in the system-equations. However, the declaration of variables or sub-models is kept away from these conditional blocks and is restricted to the unconditional header-section. Hence there is no mechanism for instance creation or removal at run-time. [1]

2.2 No Dynamic Linking

The linking of an identifier to its instance is always static in Modelica. To conveniently handle objects that are created at run time, a dynamic linking of identifiers to their instances becomes desirable. Consequently, the linking must be assigned by the use of appropriate operators. Sub-models have now to be treatable as an entity.

2.3 Nontransparent type-system

Such assignments that operate on complete model-instances also increase the emphasis on the type analysis like type-compatibility. Modelica is based on a structural type system [1] that represents a powerful and yet simple approach. Sadly, the actual type is not made evident in the language for a human reader since type-members and non-type members mix in the header-section. Also the header section itself might be partitioned in different parts. Hence it becomes hard to identify the type of sub-models just by reading its declaration. This becomes a crucial issue when objects need to be treated dynamically.

2.4 Accessing the Environment

Each model in Modelica is defined as a closed entity that cannot access by itself any outside variables. Whereas such a restriction is meaningful in most of the cases, it is inappropriate for certain tasks. One of these tasks is for instance the automatic connection of mass-holding objects to a gravity field. Modelica offers the concept of outer-models for this purpose. Unfortunately this approach is quite limited and represents not a feasible approach for more complex data-structures. At most, outer-models could be used to create pools for mutual gravitational attraction [8] or potential collisions [3]. But to enable such pools, the single-pool members had to be manually assigned to an appropriate integer-ID. This is not a convenient solution.

[1] In fact, there exists a small mechanism for conditional declaration in Modelica that is supported by Dymola, but the conditions are based upon parameters and the way it is done restricts the access on such a conditional object to connect-statements.

The dynamic creation of sub-models increases the importance of a feasible solution for this task. When objects are created dynamically, they also need to be connected to other objects in their environment. Connections to other sub-models need to be established automatically at simulation time.

2.5 Insufficient Handling of Discrete-Events

Processes for the creation, removal and handling of dynamic instances represent discrete processes. Hence a powerful support for discrete-event handling is necessary. Modelica offers hybrid extension for such modeling tasks that are inspired by the synchronous data-flow principle [5]. However, for larger systems the current implementation may lead to an computational overkill and hence more elaborated concepts are needed.

The creation and connection have to be managed by discrete events. During such a construction process, singular equation systems may temporarily occur. However, they are not meant to be evaluated. Thus, a synchronous evaluation of the complete system represents an infeasible approach for such tasks, since it can lead to the inappropriate evaluation of intermittent singular systems.

In addition, the discrete event handling is insufficiently specified in the Modelica-language definition. There is a clear lack of specification for describing what is supposed to happen exactly if one event is subsequently causing (or canceling) other events during the same point of simulation time.[2]

2.6 Tedious complexity

In the attempt to enhance the Modelica-language with regard to certain application-specific tasks, the original language has lost some of its original beauty and clarity. An increasing amount of specific elements have been added to the language that come with rather small advantages. Several of these small add-ons are potential sources for problems when structural variability is concerned. Thus, a clean-up of the language is an inevitable prerequisite for any further development in this field. Furthermore the language is subverted in daily practice by foreign elements, i.e., so-called annotations.

2.7 Summary

To express structural changes, a corresponding modeling language has to meet certain requirements. The language must support discrete events and hence support hybrid modeling, since structural changes clearly represent a discrete event. Furthermore, it

[2] This concerns for example the MultiBondLib [8] and its impulse-models. The correctness of these models cannot be proven on the basis of the language-specification. Indeed, the correct simulation of these models is bound to the specific implementation in Dymola.

must be allowed to state relations between variables or sub-models in a conditional form, so that the structure can change depending on time and state. In addition, variables and sub-models should be dynamically declarable, so that the corresponding instances can be created, handled, and deleted at run time. Modelica meets these requirements only partly and provides only very limited means for the description of such models.

2.8 MOSILAB

MOSILAB[7] offers a first approach to handling variable structure systems in a more general sense. It combines an extensive subset of Modelica with a description language for statecharts to handle the transition between different modeling modes. MOSILAB features the dynamic creation of sub-model instances, although it does so in a limited way. For us, the use of statecharts represents a practical but limited solution. However, statecharts do not integrate too well into the object-oriented and declarative framework of Modelica. Hence the complexity of the language had to be increased significantly and the beauty and clarity of the original Modelica language suffered in the process of extending the language.

3 Sol - a Derivative Language of Modelica

In attempting an enhancement of Modelica's capabilities with respect to variable structure systems, one arrives at the conclusion that a straight-forward extension of the language will not lead to a persistent solution. The introduction of additional dynamics inevitably violates some of the fundamental assumptions of the original language design and of its corresponding translation and simulation mechanisms.

Hence we have taken the decision to design a new language, optimized to suit the new set of demands. This language is called Sol. In the design process, we intend to maintain as much of the essence of Modelica as possible. To this end, we review the major strengths of Modelica:

- Modelica owns natural readable, intuitive syntax. Models can be understood even by outsiders, and beginners are enabled to quickly acquaint themselves with the language.
- The declarative, equation-based modeling approach enables the modeler to concentrate on what should be modeled, rather than forcing him or her to consider, how precisely the model is to be simulated.
- Modelica offers convenient object-oriented means for the organization of knowledge and type-generation. This makes large projects feasible and eases the knowledge transfer.
- The structural type-system of Modelica separates type-generation and implementation. Thus, even separate implementations can be compatible and exchangeable. The generic connection mechanism enables an intuitive and convenient modeling.

3.1 Sol – a New Language for Variable Structure Systems

All those considerations of the previous sections have been taken into account for the design process of Sol. The decision to design a new language enables us to take a more radical, conceptually stronger approach. Hence, Sol attempts to be a language of low complexity that still enables a high degree of expressiveness.

Like Modelica, Sol provides means for declaring synchronous, non-causal relations between variables (i.e., equations). As an extension to Modelica, we furthermore offer a convenient way for declaring asynchronous, causal transmissions from one variable (or sub-model) to another. All of these declarations can be grouped in an almost arbitrary fashion. These groups of declarations may be activated or deactivated in accordance with conditions, events or predetermined sequences.

Unlike in Modelica, also the declaration of variables and sub-models can occur at the beginning of each group or subgroup. Since these groups can be stated in a conditional form, variables and sub-models may be dynamically created and deleted at run time. Hence instance creation and deletion does not need to be stated in the (typical) imperative form. It results from the activation and deactivation of declarative groups. The dynamically created objects can be handled in an unambiguous way by the declaration of asynchronous transmissions. Identifiers can also link dynamically to an instance.

Hence systems that are expressed in Sol are described in a constructive way, where the path of construction and the corresponding interrelations might change in dependence on the current system's state or on current evaluations. Conditional declarations enable a high degree of variability in structure. The constructive approach avoids memory leaks and the description of error-prone update-processes.

The new language will be well-structured, easily readable, and intuitive to understand. The language will provide various object-oriented tools that enable the efficient handling of complex systems. The syntax and grammar of Sol is significantly stricter than the grammar of Modelica. Alternative writings have been discarded and the different sections of a model must obey a given order. This strictness unifies the writing and intends to guide towards a clear and understandable modeling style.

3.2 Example

Without going into the details concerning Sol's grammar and semantics, we provide a small, introductory example to show its potential usage. Due to Sol's similarity to Modelica and its intuitive syntax, the example should be understandable in its main functionality. In addition to classic equations Sol features copy-transmission (<<) and move-transmissions (<–). We model a simple machine, consisting of an engine that drives a fly-wheel. Two models are provided for the engine: The first model "Engine1" applies a constant torque on the flange. In the second model "Engine2", the torque is dependent on the positional state similar to a piston-engine. The

machine-model connects the engine and the fly-wheel. It contains a structural change that is reflected by a substitution of the engine-models. Initially, the fly-wheel is at rest, and the more complex engine model is used. When the speed exceeds a certain threshold, it seems appropriate to average the torque. Thus, the simpler engine-model is used instead.

```
package Rotational

    connector Flange
    interface:
          static potential Real phi;
          static flow Real t;
    end flange;

    partial model Engine
    interface:
          parameter Real meanTorque << 1;
          static Flange f;
    end Engine;

    model Engine1 extends Engine;
    implementation:
          f.t = meanTorque;
    end Engine1;

    model Engine2 extends Engine;
    implementation:
          static Real transmission;
          transmission = 1+sin(f.phi);
          f.t = meanTorque*transmission;
    end Engine2;

    model FlyWheel
    interface:
          parameter Real inertia << 1;
          static Flange f;
          static Real w;
    implementation:
          static Real z;
          w = der(f.phi);
          z = der(w);
          f.t = inertia*z;
          when initial then w=0; f.phi=0; end;
    end FlyWheel;
```

```
    model Machine
    implementation:
          static FlyWheel Wheel1{inertia << 10};
          static Boolean fast;
          if fast then
                static Engine1 E{meanTorque << 100};
                connection(E.f,Wheel1.f);
          else then
                static Engine2 E{meanTorque << 100};
                connection(E.f,Wheel1.f);
          end;

          when initial then fast << false; end;
          when Wheel1.w > 50 then fast << true; end;
    end Machine;

end Rotational;
```

The structural change is contained in the model "Machine". It declares a Boolean state-variable "fast" that determines which model to use. Please note, that the conditional if-clauses also contain declarations of sub-models. This enables a convenient, easily readable formulation of the structural change based on the current system state. There is also no need for an explicit model of the transition or manual disconnections.

The example code below presents an alternative solution for the machine-model. The identifier E is declared to be "dynamic". This means: It can be dynamically linked to any model-instance that is type-compatible with "Engine". The corresponding instances are simply declared anonymously in the conditional when-clauses. The type of a model is solely defined by its interface-section.

```
model Machine
implementation:
    static FlyWheel Wheel1{inertia << 10};
    dynamic Engine E;
    connection(E.f,Wheel1.f);
    when initial then
      E <- Engine2{meanTorque << 100};
    end;
    when Wheel1.w > 50 then
      E <- Engine1{meanTorque << 100};
    end;
end Machine;
```

This simple example contains only a very simple structural change that is basically reflected by the replacement of a single equation. Hence this could have also been modeled in Modelica, but not at this level of abstraction. The complete replacement of a model, as it is done here, can as well be used for more elaborate multi-level models.

4 Implementation and Ongoing Development

A first version of the language definition of Sol has been written down in the form of an internal report. It forms the fundamentals for a corresponding implementation that is currently under development. This implementation will be represented by an interpreter that parses the model-file, instantiates a selected model and starts simulation. In addition to its main task, the interpreter will provide various tools for the analysis of the object-hierarchy, type-structure, etc.

Whereas the pair of a compiler and a simulator is the preferred choice for high-end simulation tasks, an interpreter is an appropriate tool for research work on language design. The development process becomes much easier, faster and more flexible. Hence the development of the interpreter can proceed in parallel with a further refinement of the language. Also, new debugging techniques will be needed that can be better provided by an interpreter, since all necessary meta-information is available. Of course, any interpreter (even if it is well written) suffers from a certain computational overhead that may prevent its usage for highly demanding simulation applications. Hence an important aspect will be to sketch the development of a corresponding compiler.

4.1 Future goals

Sol is a language primarily conceived for research purposes. We want to explore the full power of a declarative modeling approach and how it can handle potential, future problem fields. Some of our goals and motivations are similar to [6], although we are coming from a different direction. The implementation of Sol will be a small and open project that should enable other researchers to test and validate their ideas with a moderate effort. The longer term goal of our research is to significantly extend Modelica's expressiveness and range of application. Furthermore, the Sol-project gives us a development-platform for technical solutions that concerns the handling of structurally changing equation systems. This includes solutions for dynamic recausalization or the dynamic handling of structural singularities.

It is not our target to immediately change the Modelica standard or to establish an alternative modeling language. Our scientific work is intended to merely offer suggestions and guidance for future development. This will primarily benefit the future development of Modelica, but our results may also prove useful to other modeling communities and researchers.

5. Conclusion

The development of a new modeling language should be a well considered step, since it incorporates a lot of effort. This does not only concern the developers of the language and the corresponding software, it includes as well the potential modelers

and users that are expected to get themselves acquainted with the new methodology. However, the continuous progress of modeling technology generates a new set of demands. This makes such a step finally inevitable.

In this workshop-paper, we offered a first glance of Sol, our new modeling language. Sol has been designed to enable the modeling of variable-structure systems using an equation-based framework. While its development is currently still at the beginning, we expect to make significant progress in the near future. In the longer term, we hope that our research will benefit Modelica's future development.

References

1. Broman, D., Fritzson, P., Furic, S.: Types in the Modelica Language. In: Proceedings of the Fifth International Modelica Conference, Vienna, Austria (2006) Vol. 1, 303-315
2. Cellier, F.E., Krebs, M.: Analysis and Simulation of Variable Structure Systems Using Bond Graphs and Inline Integration. In: Proc. ICBGM'07, 8th SCS Intl. Conf. on Bond Graph Modeling and Simulation, San Diego, CA, (2007) 29-34.
3. Elmqvist, H., Otter, M., Díaz López, D.: Collision Handling for the Modelica MultiBody Library. In: Proc. 4th International Modelica Conference, Hamburg, Germany (2006) 45-53
4. Höpler, R., Otter, M.: A Versatile C++ Toolbox for Model Based, Real Time Control Systems of Robotic Manipulators. In: Proc. of 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems, Maui, USA, (2001) 2208-2214
5. Otter, M., Elmqvist, H., Mattsson, S.E.: Hybrid Modeling in Modelica Based on the Synchronous Data Flow Principle. In: Proc. IEEE International Symposium on Computer Aided Control System Design, Hawaii. (1999) 151-157
6. Nilsson, H., Peterson J., Hudak, P.: Functional Hybrid Modeling. In: Proceedings of the 5th International Workshop on Practical Aspects of Declarative Languages, New Orleans, LA (2003) 376—390
7. Nytsch-Geusen, C. et. al.: Advanced modeling and simulation techniques in MOSILAB: A system development case study. In: Proceedings of the Fifth International Modelica Conference, Vienna, Austria (2006) Vol. 1, 63-71
8. Zimmer, D., Cellier, F.E., The Modelica Multi-bond Graph Library. In: Proc. 5th Intl. Modelica Conference, Vienna, Austria (2006) Vol. 2, 559-568