Proposal for A Context-oriented Modelica Contributing to Variable Structure Systems

Zizhe Wang[1,2*] Manuel Krombholz[2*] Uwe Aßmann[2] John Tinnerholm[3] Christian Gutsche[1,2]
Volodymyr Prokopets[2] Sebastian Götz[2]

[*]Co-first author
[1]Boysen-TU Dresden-Research Training Group, Dresden, Germany
[2]Software Technology Group, Technische Universität Dresden, Germany
[3]Department of Computer and Information Science, Linköping University, Sweden
zizhe.wang, manuel.krombholz, uwe.assmann, christian.gutsche,
volodymyr.prokopets, sebastian.goetz1@tu-dresden.de
john.tinnerholm@liu.se

Abstract

Context-aware systems are widespread in our daily lives, but modeling languages that address the notion of context are rare. Variable structure systems (VSS) allow for structural and behavioral changes in physical models at runtime (while the simulation is running) based on different situations. It is desirable to explicitly describe under which contextual situation a specific variant of the simulation model should be used and how to implement the switching between these variants at runtime. In this case, contexts could be used to control the variability of context-aware systems. Equation-based modeling languages are suitable for modeling complex multi-domain, multi-physical systems, and among them, Modelica is the state-of-the-art. Unfortunately, the capabilities for VSS in Modelica are strongly limited. As a result, several frameworks have been proposed to address this problem by supporting different VSS types. However, it remains unclear which framework contributes to which VSS type. Furthermore, approaches have been developed to support VSS, but none can explicitly describe contexts and their transitions. In this work, we first introduce VSS and its different types. Then, we provide an overview of which framework targets which VSS type. Finally, we propose a new language extension based on Modelica, ContextModelica, that provides semantics for the direct context definition, enabling the use of context to control and manage variability.

Keywords: modeling and simulation, Modelica, variable structure systems, context, context-oriented programming, ContextModelica

1 Introduction

1.1 Context-aware systems

Context-aware systems are widely present in different aspects of our daily lives. According to Dey, Abowd, et al. 2000, a context is "any information that can be used to characterize the situation of an entity. An entity is a person, place, or object considered relevant to the interaction between a user and an application, including the user and application themselves". Many context-aware systems operate according to system contexts (e.g., "a robot should stop working when a human enters its operation area", "an iPhone will make emergency calls if a car crash has been detected"). Different context-oriented techniques have been developed to enhance context-aware systems, including Context-Oriented Programming (Hirschfeld, Costanza, and Nierstrasz 2008), commonly referred to as COP. Elyasaf, Cardozo, and Sturm 2023 and Elyasaf and Sturm 2023 state "Although COP languages have existed for over 15 years, they are still very limited for developing context-aware systems. Also, modeling languages that address the notion of context are rare." Thus, how the idea of COP could be implemented in equation-based modeling languages, such as Modelica, remains a research question.

1.2 Variable Structure Systems and Modelica

Utkin 1977 introduced variable structure systems (VSS), which consist of continuous subsystems with a proper switching logic and enable dynamic control of simulation systems. In real applications, certain conditions, such as contexts (Elyasaf and Sturm 2022), can be used to control the variability (different modes). Modes refer to different states; different modes correspond to different models defined by distinct equation systems.

Figure 1 shows a minimal example. On a sunny day (*Context* = *Sunny*), solar radiation is present, and the mode "*Solar Power*" is activated. This mode engages the corresponding equation system, which represents the solar panels. Thus, in the *Sunny* context, the solar panels are activated and begin producing electricity from solar energy. In the evening (*Context* = *Night*), solar radiation is absent, and the mode "*Standby*" with its corresponding equation system is activated (while other modes and their equation systems are deactivated). The equation system for this mode represents a physical state where the solar panels

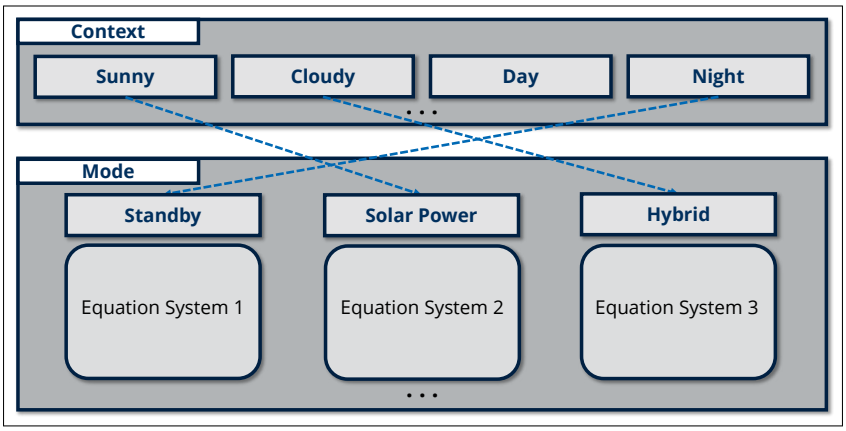

Figure 1. A minimal example of using contexts to control different modes.

are inactive. In realistic applications, multiple systems can be switched simultaneously. All of this occurs at runtime (simulation time); there is no need to power off the system, switch modes, and then re-initialize and restart the simulation. Typical application fields where VSS can be beneficial include circuit switching, mechanical elements breaking apart, systems with clutches, different rocket stages, and robot reconfigurations.

Modern modeling environments handle complex physical systems using equation-based modeling languages, also known as acausal modeling languages. The Modelica language (Fritzson and Engelson 1998) (Modelica, for short) is the state-of-the-art example, widely used in various industries like energy grids (Senkel et al. 2021) and building systems (Wetter et al. 2014). However, like most equation-based modeling languages, the possibilities for VSS in Modelica (current version 3.6) are limited. Only a few frameworks have been designed to support VSS in Modelica, and in most cases, switching modes at runtime fails. Zimmer 2010 attributes these limitations to the static treatment of the differential-algebraic equations (DAEs) and the lack of expressiveness in the Modelica language.

A classic example of VSS is the "breaking pendulum" (Figure 2) which can be described as follows: a ball attached to a string moves as a pendulum initially (mode 1). After a few seconds, the string breaks, and the ball moves as if in free fall (mode 2). This example includes two modes, each corresponding to a different model: one describes the pendulum (Listing 2), and the other describes the free fall of the ball (Listing 3). Mode switching is triggered by *time*. It is important to note that the two models have different equation systems. Classical Modelica environments, such as OpenModelica (Fritzson, Pop, et al. 2022) and Dymola (Elmqvist 1979), which are based on the current Modelica specification (Modelica Association 2023), cannot handle this situation effectively. The simulation will fail at the moment when the modes are switched. Typically, different modes are modeled and simulated separately. Ideally, developers would model and integrate different modes within a single model.

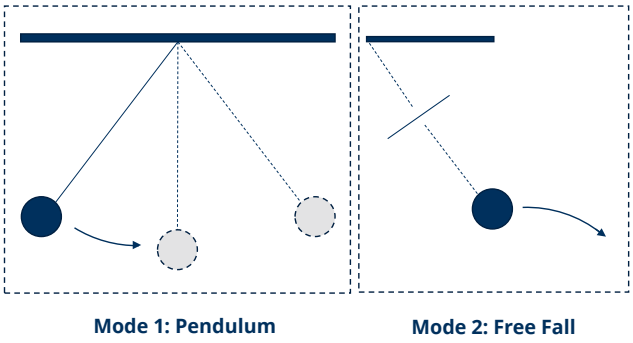

Figure 2. Two modes of the breaking pendulum.

1.3 Problem Statement and Research Objectives

During literature research, two main problems related to VSS in Modelica have been identified:

1. Despite various frameworks targeting VSS in Modelica (Table 1) and Modelica-like environments, it remains unclear which solution contributes to which VSS type. A detailed overview is lacking.

2. Enabling contexts significantly impacts the control of variability and the realization of context-aware systems. However, the idea of COP has not yet been implemented in Modelica. How to introduce contexts in Modelica remains an interesting research question.

This work aims to address these two problems. The main goals are:

1. To provide a clear classification of VSS in Modelica and an overview of frameworks supporting their VSS types.

2. To propose the extension *ContextModelica* that introduces contexts into Modelica.

 10.3384/ECP20753 DOI

Table 1. Frameworks targeting VSS in Modelica and other equation-based modeling environments.

Frameworks
Mosilab (Nytsch-Geusen et al. 2005)
Sol (Zimmer 2010)
Hydra (Giorgidze 2012)
Modelyze (Broman and Siek 2012)
DySMo (Möckel, Mehlhase, and Nytsch-Geusen 2015)
MoVasE (Esperon, Mehlhase, and Karbe 2015)
PyVSM (Stüber 2017)
Modia.jl (Elmqvist, Neumayr, and Otter 2018)
OM.jl (Tinnerholm, Pop, Sjölund, et al. 2020)
ModelingToolkit.jl (Ma et al. 2021)

1.4 Structure of the Work

Section 2 provides a detailed explanation and classification of VSS in Modelica. Section 3 summarizes various frameworks designed to support different VSS types in Modelica or Modelica-like environments, offering an overview to understand which framework addresses which specific VSS problem. In Section 4, we propose ContextModelica, developed based on OpenModelica.jl, in short, OM.jl[1] (Tinnerholm, Pop, Sjölund, et al. 2020). Our extension combines Modelica with the concept of context from the language engineering field. We demonstrate ContextModelica with an example and discuss the current challenges. This section also explores the potential benefits of integrating context-aware features into existing Modelica models. Finally, Section 5 presents the conclusions and an outlook, including discussions and suggestions for future research.

2 VSS in Modelica

To provide an extension that enables modeling and managing VSS using contexts in Modelica, the first step is to understand what VSS are. Definitions of VSS vary slightly across different domains in the literature. VSS were first introduced by Utkin 1977. Mehlhase 2015 offers an overview of publications with definitions related to VSS. In short, VSS can be summarized as "structural change during runtime (simulation time)". In Modelica, VSS correspond to the switching of equation systems based on different situations while the simulation is running. However, different types of structural change during runtime exist, and Modelica supports only some of them in a limited way. Consequently, various frameworks have been designed to enhance VSS possibilities in Modelica and Modelica-like environments. Unfortunately, since the types of structural changes during runtime have not been discussed in detail, it remains unclear, which framework addresses which specific VSS type.

[1]https://github.com/JKRT/OM.jl (A Modelica compiler written in Julia)

To the best of our knowledge, **variables** and **differential index**[2] have the most impact on realizing VSS in Modelica. In general, three different types can be distinguished based on these two factors:

1. **Two modes share the same variables and differential index.** Thus, the structural change does not introduce new variables, and the differential index remains unchanged.

2. **Two modes have different variables but the same differential index.** In this case, the structural change introduces new variables and corresponding equations, while the differential index stays the same.

3. **Two modes have the same variables but different differential indices.** Here, the structural change involves a change in the differential index.

At this point, some issues related to VSS arise in Modelica (Benveniste, Caillaud, et al. 2019). In most cases, the simulation fails when switching from one mode to another, primarily because Modelica is static and the compiler cannot handle types 2 and 3 at runtime.

Regarding type 2 where each mode contains a different set of variables, there are several sub-types. The number of variables may either change or remain the same during the mode transition. For simplicity, this work does not specify different sub-types of variables.

3 State of the Art

This section provides an overview of the frameworks for supporting different VSS types in Modelica and other equation-based modeling environments (most of them are Modelica-like). Table 2 summarizes the applicability of approaches for different VSS types. All approaches can be used for type 1.

Mosilab (Nytsch-Geusen et al. 2005) uses a Modelica extension to describe the models and transitions through a state chart.

Mosilab supports types 1 and 2 but does not support type 3, as the environment only simulates index-0 models and lacks an index-reduction mechanism.

Sol (Zimmer 2010) is an experimental language designed as a proof of concept to support variable-structure models using dynamic casualization.

Although Sol is similar to Modelica, it is a separate language. It enables the modeling of VSS with *SolSim* and allows changes to the differential index.

[2]For a general differential algebraic equation (DAE) $F(t,x,x') = 0$, the differential index is defined as "the minimum number of differentiations required to translate the DAE system into a system of the ordinary differential equations (ODEs)" (Campbell and Gear 1995) (Benveniste, Bourke, et al. 2014). Thus, ODEs have a differential index of 0.

Hydra (Giorgidze 2012) is an embedded acausal modeling language implemented in Haskell according to the paradigm of functional hybrid modeling. Hydra lacks the object-oriented characteristic present in modeling languages such as Modelica.

Modelyze (Modeling Kernel Language) (Broman and Siek 2012) is a host language designed for embedding equation-based DSL based on gradual typing. Modelyze has been developed with OCaml.

Dymola extensions Elmqvist, Mattsson, and Otter 2014 and Mattsson, Otter, and Elmquvist 2015 present extensions of Dymola to enable the possibility of VSS. In the first work VSS with varying DAE index is not supported, the second work extends the Pantelides algorithm (Pantelides 1988) and allows VSS with varying DAE index. These extensions have limited functionality. They have only been tested with simple examples. Because of this, these extensions have not been implemented in the latest stable release of Dymola yet (as of May 2024).

DySMo (Dynamic Structure Modeling) (Mehlhase 2015) is a Python application that enables the simulation of VSS. A case study by Möckel, Mehlhase, and Nytsch-Geusen 2015 demonstrated the use of DySMo in the context of building simulation.

DySMo is a script-based approach designed for simulating VSS rather than modeling them. In this approach, different models are simulated separately, and their results are then integrated using Python.

MoVasE (Modelica Variable-structure Editor) (Esperon, Mehlhase, and Karbe 2015) enables structural changes to models by defining conditional exchanges externally.

Compared to DySMo, MoVasE has the advantage of not requiring manual creation and maintenance of all modes and transitions. However, this approach still has limitations in terms of the dynamic addition and removal of components.

PyVSM (Python Variable-structure Model) (Stüber 2017) is another script-based approach using Dymola's Python interface. The idea is the same as in DySMo: Using Modelica for simulating different modes and Python for switching between them.

Modia.jl (Elmqvist, Neumayr, and Otter 2018) is a Modelica-like software written in Julia. It has been initiated by the inventor of Dymola. After several attempts to support VSS in Dymola, as discussed in Elmqvist, Mattsson, and Otter 2014 and Mattsson, Otter, and Elmquvist 2015, the authors explored Julia's potential in modeling. Modia.jl utilizes predefined acausal components, as described in Neumayr and Otter 2023.

OM.jl OpenModelica.jl (Tinnerholm, Pop, Sjölund, et al. 2020) is a Modelica compiler written in Julia, developed by the OpenModelica development team from Linköping, Sweden. Leveraging Julia's just-in-time (JIT) compilation and multi-dispatch features, OM.jl supports modeling VSS. It can also connect ModelingToolkit.jl with Modelica (Tinnerholm, Pop, Heuermann, et al. 2021).

ModelingToolkit.jl (Ma et al. 2021)[3] is a Julia package for modeling and simulation that integrates Julia's ecosystem with the modeling. Inspired by Modelica, it features a Modelica-like syntax. Compared to Modelica, ModelingToolkit.jl supports not only ODEs and DAEs, but also partial differential equations (PDEs), stochastic differential equations (SDEs), and other types of equation systems. Like Modia.jl and OM.jl, ModelingToolkit.jl supports various VSS types due to Julia's capabilities.

Table 2. Overview of Modelica-based and Modelica-like frameworks for different VSS types. ✓ indicates that the approach supports this VSS type, while ✗ indicates that it does not.

VSS Types	*Applicability of Approaches*
Type 1	All approaches (✓)
Type 2	Standard Modelica (✗) Mosilab (✓, but only index 0) Sol (✓) Dymola extensions (✓) DySMo (✓) MoVasE (unknown, lack of literature) PyVSM (✓) Modia.jl (✓) OM.jl (✓) ModelingToolkit.jl (✓)
Type 3	Standard Modelica(✗) Mosilab (✗) Sol (✓) Dymola extensions (✓) DySMo (✓) MoVasE (unknown, lack of literature) PyVSM (✓) Modia.jl (✓) OM.jl (✓) ModelingToolkit.jl (✓)

Despite some approaches supporting all VSS types, **none of the above approaches support the explicit specification of contexts and their transitions.**

[3] https://docs.sciml.ai/ModelingToolkit/dev

4 ContextModelica

The previous section introduced different approaches to support VSS in Modelica. However, none of these approaches provide semantics to support contexts and context management within Modelica for modeling and managing context-aware systems. Therefore, we propose the *ContextModelica* - an extension of the Modelica language based on OM.jl, which already includes capabilities for supporting structural transitions between variants in Modelica.

This section will examine the concepts behind this extension in more detail and describes how it can be applied to manage VSS, along with an illustrative example. Finally, we will discuss the current challenges and limitations of the extension.

4.1 Units of Variability

In a software language, variability relies on *variation points* (Webber and Gomaa 2004). The Variation Point Model (VPM) is designed to model variation points contained in reusable software components (Webber and Gomaa 2004). Variation points are the units of variation in a specification of a program. For Modelica, several kinds of variation points can be considered, some of which are intended by the language designers.

Class and subclass Modelica is a *static* object-oriented language in which classes can be specialized by subclasses. These subclasses can be defined in variations. Therefore, a class is a static variation point in Modelica, and it is common to replace a class with one of the members of its transitively defined derived subclasses.

Equation block An equation block defines a set of variables or derivatives, constituting the provided interface of the block. In Modelica, equation blocks can be guarded by `if/else` and `when` statements, allowing them to be dynamically varied (dynamic variation point).

Equation A single equation can also be a variation point. It is a special case of a block variation point.

As discussed, modes in VSS may differ in variables and the differential index. Modes relate to variation points in that these constraints about variables and the differential index must hold also for all variants of a Modelica variation point. This means that for any pointwise variation, the VSS types 1-3 can be distinguished. For instance, a variation point of type 1 can be a class, block, or equational variation point.

For a *class variation point*, VSS type 1 is the simplest type, where polymorphism of the class resolves the transition to another subclass. At runtime, the subclass can be varied by wrapping all variant subclasses in a simple case expression. In Modelica, polymorphism is not available because subclasses must be selected statically. A *block variation point* of VSS type 1 can also be handled in Modelica if the block is encapsulated by a case expression. All frameworks discussed in section 3 offer dynamic block variation. Usually, an *equational variation point* of VSS type 1 can also be managed because one equation is a simple equational block.

We will demonstrate later how ContextModelica can be employed for the variation points described above.

4.2 Context

While addressing the lack of VSS manageability in Modelica, one potential solution is to introduce a language concept called a *context*. Contexts are common used in software development to separate concerns (Hirschfeld, Costanza, and Nierstrasz 2008). By integrating contexts into Modelica, we can achieve better code structure and improved manageability of VSS.

To this end, we have extended the Modelica language to include this concept and thus created ContextModelica. This extension introduces two new semantics, as shown in Listing 1. First, all modes can be listed in a separate section using the keyword "context", with each mode associated with corresponding condition. Second, the new semantics allow for the addition of multiple equation systems, with each system labeled by the corresponding context. This means that the equation system represents the model or mode when the context is active. Additionally, the set of contexts must include an initial state, which is the mode that is active at the start of the simulation.

The advantage of mapping contexts to their applicable conditions is that developers no longer need to manage the resulting transitions between contexts. This separation of concerns leads to cleaner and more readable code, particularly when compared to the use of `if/else` and `when` statements in large-scale systems.

Listing 1. Semantics in ContextModelica.

```
model ExampleModel
    /*parameters & variables*/

    equation on initial (ContextA)
        /*corresponding equations*/

    equation on ContextB
        /*corresponding equations*/

    equation on ContextC
        /*corresponding equations*/

        ...

    context
        initial on /*condition*/;
        ContextB on /*condition*/;
        ContextC on /*condition*/;
        ...
    end context;

end ExampleModel;
```

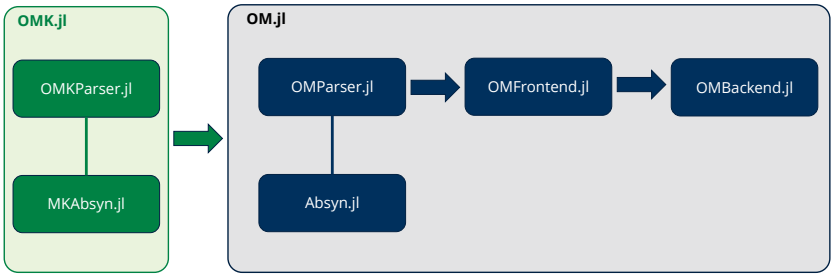

Figure 3. Structure of ContextModelica. The blue section represents the original OM.jl, while the green section indicates the added preprocessor.

4.3 How It Works

ContextModelica[4] is implemented as a language extension of Modelica in Julia (OM.jl). It benefits from the structural transitions available in OM.jl, which can be used to construct state machines in Modelica. More precisely, the defined contexts and their associated conditions from a given ContextModelica model are translated into a *context transition automaton* comprising states and transitions, representing the possible changes of the contexts. This results in *n*(n-1)* state transitions, where *n* is the number of existing states. The context transition automaton can be realized through the dynamic recompilation features of OM.jl.

The structure of ContextModelica is illustrated in Figure 3. *OMK.jl* functions as a preprocessor for OM.jl and was developed by reusing components of OM.jl, including the ANTLR parser generator (Parr and Quong 1995) and the abstract syntax tree (AST) module. Both have been slightly modified to support the new semantics introduced in ContextModelica, specifically the definition of contexts with their corresponding conditions and the equation systems that can be tagged with context labels. In addition to these modules, we added a code generator backed by some OpenModelica packages. It traverses the AST constructed by the parser and then generates the corresponding state machine using the syntax of structural transitions provided by OM.jl. Therefore the code generator gathers context labels, active conditions, and the associated equation sets, creating sub-models within a larger model. Afterward, the transitions supported by OM.jl are inserted. The resulting state machine is an undirected graph where every state has a transition to every other state. The output can then be passed to OM.jl, which generates the corresponding Julia code for further simulation.

ContextModelica inherits the ability of OM.jl to support the change of differential index, thus supporting type 3. Type 2 is currently not supported because all variables and parameters share a common set. This is due to the focus on varying the actual behavior in the individual contexts, which is primarily determined by the equation systems. Future modifications should allow separate definitions for local variables and parameters. In conclusion, ContextModelica supports two VSS types: types 1 and 3.

[4]https://github.com/dev-manuel/OMK.jl

4.4 Example

We demonstrate the proposed ContextModelica using the classical "breaking pendulum" model, as shown in Figure 2. Listing 2 and Listing 3 show the Modelica models for the "Pendulum" mode and the "FreeFall" mode respectively. With the classical Modelica software which has limited functionality of VSS the developers need to model and simulate them separately. In ContextModelica, these two models can be integrated into one model as VSS with two modes, as Listing 4 shows. Two different equation sets together with switch mechanisms will be defined in the model. The outcome of the preprocessor is shown in Listing 5. It includes the whole context transition automaton containing the models and transitions required by OM.jl for further simulation. This model corresponds to VSS type 3 because the differential index of the "Pendulum" and "FreeFall" modes are different. The result is shown in Figure 4.

Listing 2. A pendulum model written in Modelica.

```
model Pendulum
    parameter Real g = 9.81;
    parameter Real L = sqrt(200);
    Real x(start = 10);
    Real y(start = 10);
    Real vx; Real vy;
    Real phi(start=1.0); Real phid;
    equation
        der(phi) = phid;
        der(x) = vx;
        der(y) = vy;
        x = L * sin(phi);
        y = -L * cos(phi);
        der(phid) = -g / L * sin(phi);
end Pendulum;
```

Listing 3. A free fall model written in Modelica.

```
model FreeFall
    Real x; Real y; Real vx; Real vy;
    parameter Real g = 9.81;
    parameter Real vx0 = 0.0;
    equation
        der(x) = vx;
        der(y) = vy;
        der(vx) = vx0;
        der(vy) = -g;
end FreeFall;
```

Listing 4. Syntax for modeling the "Breaking Pendulum" model in ContextModelica. This model corresponds to VSS type 3.

```
model BreakingPendulum
    Real x; Real y; Real vx; Real vy;
    Real phi(start=1.0); Real phid;
    parameter Real g = 9.81;
    parameter Real vx0 = 0.0;
    parameter Real L = sqrt(200);

    //context = Initial (Pendulum)
    equation on initial
        der(phi) = phid;
        der(x) = vx;
        der(y) = vy;
        x = L * sin(phi);
        y = -L * cos(phi);
        der(phid) = -g / L * sin(phi);
    //context = FreeFall
    equation on FreeFall
        der(x) = vx;
        der(y) = vy;
        der(vx) = vx0;
        der(vy) = -g;

    //switch of contexts
    context
        initial on t < 5;
        FreeFall on t >= 5;
    end context;

end BreakingPendulum;
```

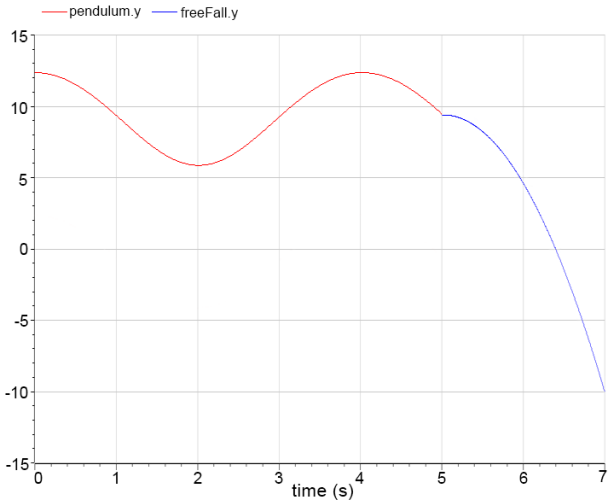

Figure 4. Simulation result of the "breaking pendulum" model with ContextModelica.

Compare the same "breaking pendulum" model implemented in ContextModelica (Listing 4) and OM.jl (Listing 6). ContextModelica enables the explicit definition of contexts directly while defining the corresponding equation systems for each mode, eliminating the need to define structural modes separately. The transition process is also simplified. In OM.jl, the transition process must be defined with a separate equation system, while in ContextModelica, this is unnecessary. The explicit definition of contexts in ContextModelica results in a cleaner structure and readable code for realizing and managing VSS, especially in large context-aware systems.

Listing 5. Transpiled model compatible with OM.jl.

```
model BreakingPendulum
    // BreakingPendulum = BP
    structuralmode
        BP__Context_Initial
        bP__Context_Initial_instance;
    structuralmode
        BP_FreeFall
        bP_FreeFall_instance;

    Real x; Real y; Real vx; Real vy;
    Real phi(start=1.0); Real phid;
    parameter Real g=9.81;
    parameter Real vx0=0.0;
    parameter Real L = sqrt(200);

    model BP__Context_Initial
        equation
            /*equation set*/
    end BP__Context_Initial;

    model BP_FreeFall
        equation
            /*equation set*/
    end BP_FreeFall;

    equation
    initialStructuralState(
        bP__Context_Initial_instance);
    structuralTransition(
        bP__Context_Initial_instance,
        bP_FreeFall_instance,
        t >= 5);
end BreakingPendulum;
```

Listing 6. Syntax of "Breaking Pendulum" model in OM.jl

```
model BreakingPendulum

    model FreeFall
        /*parameters*/
        /*variables*/
    equation
        /*equations*/
    end FreeFall;

    model Pendulum
        /*parameters*/
        /*variables*/
    equation
        /*equations*/
    end Pendulum;

structuralmode Pendulum pendulum;
structuralmode FreeFall freeFall;

equation
    initialStructuralState(pendulum);
    structuralTransition(
        pendulum, freeFall,
        t >= 5
        );

end BreakingPendulum;
```

The example also shows how ContextModelica can be deployed to the block and the equational variation points. Each equation block, or single equation in specific cases, will be varied by switching on/off different contexts/-modes. In conclusion, ContextModelica supports VSS types 1 and 3 as well as block and equational variation points, as summarized in Table 3.

Table 3. Supported VSS types and variation points of ContextModelica.

Supported VSS types	*Supported variation points*
VSS type 1 ✓	Class and subclass
VSS type 2	Equation Block ✓
VSS type 3 ✓	Equation ✓

4.5 Challenges

One challenge is synchronizing variable values when transitioning from one mode to another. Currently, all variables and parameters in a model must be defined as global ones, making them valid across all modes. For example, in Listing 4 the variables `phi` and `phid` are only used in the first mode (Pendulum), which results in redundant variables for the second mode (FreeFall). This limitation means that specific variables and parameters cannot be defined within their corresponding modes. This characteristic leads to the lack of support for VSS type 2 and this may negatively impact the performance, especially in large systems. Another challenge is that, currently, the OM.jl version only supports structural transitions in the top-level model of a program. As a result, one of the limitations is that only the top-level model of a program can have user-defined contexts. This means the use of contexts in submodels is not supported at the moment (e.g. different contexts can be defined under the "BreakingPendulum" model, but no contexts could be defined under the "FreeFall" submodel). Still, the number of modes/contexts in the top-level model is not limited. Because of this, ContextModelica does not fully support class/subclass variation point. Another challenge is the overlap of transition conditions. If two conditions can be evaluated to be true at the same time, unexpected behavior may occur because the transition is not deterministic. For now, the developers need to ensure that the conditions are mutually exclusive to avoid such issues.

5 Conclusion and Future Work

To fully demonstrate and explain the restricted VSS feature in Modelica, we have discussed the background and explained how combining COP with Modelica can help manage variability in context-aware systems. Modeling variability using contexts reveals the switch mechanisms, aiding developers in understanding and maintaining models more effectively. Following this, we presented a classification of VSS types as well as a detailed overview of various frameworks designed to support VSS in Modelica or Modelica-like environments, covering different VSS types. Unfortunately, none of these frameworks support the explicit specification of contexts, making it difficult to manage variability in context-aware environments. Therefore, we proposed the ContextModelica, a context-oriented extension of Modelica *ContextModelica* with easy-to-understand semantics. This approach also avoids the complexities of using `if/else` and `when` statements in large-scale systems. ContextModelica supports VSS types 1 and 3, as well as "equation block" and "equation" variation points. **To our knowledge, the proposed ContextModelica is the first approach that introduces the concept of context and COP into Modelica.** It extends the Modelica language with the explicit specification of context, providing a novel solution to model and manage variability in context-aware systems.

Note that the VSS can be quite complex, and this complexity must be addressed in future work. On one hand, contexts can either be mutually exclusive or overlapping, which adds complexity to our implementation. We need to carefully consider and address these scenarios to ensure that our system can handle both exclusive and non-exclusive contexts effectively. On the other hand, in our example, we only covered contexts that are time-relevant. However, there can also be time-irrelevant contexts. For instance, after the "FreeFall" mode, when the ball hits the ground and switches to the "BouncingBall" mode, it is challenging to define the exact moment the ball hits the ground. In such cases, time-irrelevant contexts are useful, e.g., when the ball hits the ground and its acceleration vectors changes direction, at this moment, the third mode "BouncingBall" is activated, as shown in Listing 7. While ContextModelica can technically handle this scenario, we do not consider it a verified example without thorough testing. More tests are needed to explore potential issues that might arise in such cases.

Listing 7. Syntax for adding the "BouncingBall" mode.

```
model BreakingPendulum
    /*parameters*/
    /*variables*/

    equation on initial
        /*equations*/

    equation on FreeFall
        /*equations*/

    equation on BouncingBall
        /*equations*/

    //switch of contexts
    context
        initial on t < 5;
        FreeFall on t >= 5;
        BouncingBall on vy < 0;
    end context;

end BreakingPendulum;
```


10.3384/ECP20753 DOI

Another complexity of VSS is the concept of **unilateral constraints**, as explored in the works of Ch Glocker and Pfeiffer 1992, Friedrich Pfeiffer and Christoph Glocker 2000, Enge and Maißer 2005, and Enge-Rosenblatt 2017, as well as in the PhD theses of Christoph Glocker 1995 and Enge 2005, where the switching between modes is driven by these constraints. For example, this occurs when a normal "Pendulum" mode transitions to a string-bound free-flying mode Figure 5, or in switching diodes used in power electronics. In the first example, defining the exact point of transition is difficult, unlike in a scenario involving a pendulum string breaking, which can be clearly identified. The transitions in the second example differ depending on the direction of the switching.

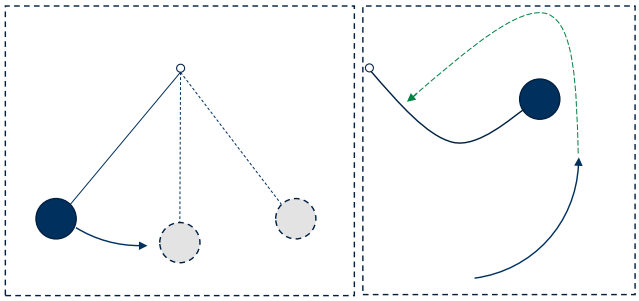

Figure 5. Transition from the pendulum mode to the string-bound free-flying mode.

The two challenges discussed in Section 4.5 are also crucial for future work. Firstly, VSS type 2 is not supported since all variables and parameters should be defined as global variables and parameters, this may lead to redundancy of variables and thus performance issues, especially in large systems. Secondly, only contexts in the top-level model are supported. It would be more practical to also enable defining and using contexts in submodels. This will also allow ContextModelica to support the *class and subclass* variation point. It should be noted that OM.jl supports both structural transitions and recompilation constructs. However, currently, ContextModelica only supports structural transitions. Implementing the recompilation constructs in ContextModelica would help solve these two challenges and improve the performance significantly. Listing 8 shows an example of recompilation constructs used in OM.jl for the "breaking pendulum" model[5]. In this example, variables and parameters for different submodels can be defined separately in the submodels rather than as global variables and parameters. Implementing recompilation constructs to support VSS type 2 and nested contexts in submodels should be considered for future development. Furthermore, more practical and industry-oriented examples should be examined using ContextModelica.

Listing 8. Syntax using recompilation constructs in OM.jl.

```
model BreakingPendulum

    model FreeFall
    /*parameters & variables*/
    equation
        /*equations*/
    end FreeFall;

    model Pendulum
    /*parameters & variables*/
    equation
        /*equations*/
    end Pendulum;

parameter Boolean breaks = false;
FreeFall freeFall if breaks;
Pendulum pendulum if not breaks;

equation
    when 5.0 <= time then
        recompilation(breaks, true);
    end when;

end BreakingPendulum;
```

Acknowledgements

The authors would like to thank the Boysen–TU Dresden–Research Training Group for the financial and general support that has made this contribution possible. The Research Training Group is co-financed by the Friedrich and Elisabeth Boysen Foundation and TU Dresden. This work is also financially supported by the German Research Foundation (Project No. 453596084 - SFB/TRR 339). The authors would also like to thank Prof. Peter Fritzson and Dr. Adrian Pop from Linköping University for their valuable discussions, as well as Anastasiia Korolenko for her contribution to the test suite.

References

Benveniste, Albert, Timothy Bourke, et al. (2014). *On the index of multi-mode DAE Systems (also called Hybrid DAE Systems)*. Research Report. Inria.

Benveniste, Albert, Benoıt Caillaud, et al. (2019). "Multi-mode DAE models-challenges, theory and implementation". In: *Computing and Software Science: state of the Art and Perspectives*, pp. 283–310.

Broman, David and Jeremy G Siek (2012). "Modelyze: a gradually typed host language for embedding equation-based modeling languages". In: *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2012-173*.

Campbell, Stephen L and C William Gear (1995). "The index of general nonlinear DAEs". In: *Numerische Mathematik* 72.2, pp. 173–196.

Dey, Anind K, Gregory D Abowd, et al. (2000). "The context toolkit: Aiding the development of context-aware applications". In: *Workshop on Software Engineering for wearable and pervasive computing*. University of Washington Seatle, pp. 431–441.

[5] https://github.com/JKRT/OM.jl/tree/master/test/Models/VSS

Elmqvist, Hilding (1979). "DYMOLA-a structured model language for large continuous systems". In: Summer Computer Simulation Conference, Toronto, Canada.

Elmqvist, Hilding, Sven Erik Mattsson, and Martin Otter (2014). "Modelica extensions for multi-mode DAE systems". In: *Proceedings of the 10th international Modelica conference*. 96. Linköping University Electronic Press Linköping, Sweden, pp. 183–193.

Elmqvist, Hilding, Andrea Neumayr, and Martin Otter (2018). "Modia-dynamic modeling and simulation with julia". In: Juliacon, University College London, UK.

Elyasaf, Achiya, Nicolás Cardozo, and Arnon Sturm (2023). "A framework for analyzing context-oriented programming languages". In: *Journal of Systems and Software* 198, p. 111614.

Elyasaf, Achiya and Arnon Sturm (2022). "Modeling Context-aware Systems: A Conceptualized Framework." In: *MODELSWARD*, pp. 26–35.

Elyasaf, Achiya and Arnon Sturm (2023). "A Framework for Analyzing Modeling Languages for Context-Aware Systems". In: *SN Computer Science* 4.2, p. 149.

Enge, Olaf (2005). "Analyse und Synthese elektromechanischer Systeme". PhD thesis. Technische Universitaet Chemnitz (Germany).

Enge, Olaf and Peter Maißer (2005). "Modelling electromechanical systems with electrical switching components using the linear complementarity problem". In: *Multibody System Dynamics* 13, pp. 421–445.

Enge-Rosenblatt, Olaf (2017). "Equation-based modelling and simulation of hybrid systems". In: *Proceedings of the 8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, pp. 27–36.

Esperon, Daniel Gomez, Alexandra Mehlhase, and Thomas Karbe (2015). "Appending variable-structure to modelica models (WIP)". In: *Proceedings of the Conference on Summer Computer Simulation*, pp. 1–6.

Fritzson, Peter and Vadim Engelson (1998). "Modelica—A unified object-oriented language for system modeling and simulation". In: *ECOOP'98—Object-Oriented Programming: 12th European Conference Brussels, Belgium, July 20–24, 1998 Proceedings 12*. Springer, pp. 67–90.

Fritzson, Peter, Adrian Pop, et al. (2022). "The OpenModelica integrated environment for modeling, simulation, and model-based development". In: Mic.

Giorgidze, George (2012). "First-class models: On a noncausal language for higher-order and structurally dynamic modelling and simulation". PhD thesis. University of Nottingham (England).

Glocker, Ch and F Pfeiffer (1992). "Dynamical systems with unilateral contacts". In: *Nonlinear Dynamics* 3, pp. 245–259.

Glocker, Christoph (1995). "Dynamik von Starrkörpersystemen mit Reibung und Stößen". PhD thesis. Technische Universitaet Muenchen (Germany).

Hirschfeld, Robert, Pascal Costanza, and Oscar Nierstrasz (2008). "Context-oriented programming". In: *Journal of Object technology* 7.3, pp. 125–151.

Ma, Yingbo et al. (2021). "Modelingtoolkit: A composable graph transformation system for equation-based modeling". In: *arXiv preprint arXiv:2103.05244*.

Mattsson, Sven Erik, Martin Otter, and Hilding Elmquvist (2015). "Multi-mode DAE systems with varying index". In: *Proceedings of the 11th International Modelica Conference*, pp. 89–98.

Mehlhase, Alexandra (2015). "Konzepte für die Modellierung und Simulation strukturvariabler Modelle". PhD thesis. Technische Universitaet Berlin (Germany).

Möckel, Jens, Alexandra Mehlhase, and Christoph Nytsch-Geusen (2015). "Exploiting variable-structure models in the context of building simulations within Modelica". In: *Proceedings of BS2015. International Building Performance Simulation Association*.

Modelica Association (2023-03). *Modelica – A Unified Object-Oriented Language for Systems Modeling. Language Specification Version 3.6*. Tech. rep. Linköping: Modelica Association. URL: https://specification.modelica.org/maint/3.6/MLS.pdf.

Neumayr, Andrea and Martin Otter (2023). "Variable Structure System Simulation via Predefined Acausal Components". In: *Proceedings of the 15th International Modelica Conference*.

Nytsch-Geusen, Christoph et al. (2005). "MOSILAB: Development of a Modelica based generic simulation tool supporting model structural dynamics". In: *Proceedings of the 4th International Modelica Conference TU Hamburg-Harburg*. Vol. 2. Citeseer.

Pantelides, Constantinos C. (1988). "The Consistent Initialization of Differential-Algebraic Systems". In: *SIAM Journal on Scientific and Statistical Computing* 9.2, pp. 213–231. DOI: 10.1137/0909014.

Parr, Terence J. and Russell W. Quong (1995). "ANTLR: A predicated-LL (k) parser generator". In: *Software: Practice and Experience* 25.7, pp. 789–810.

Pfeiffer, Friedrich and Christoph Glocker (2000). *Multibody dynamics with unilateral contacts*. Vol. 421. Springer Science & Business Media.

Senkel, Anne et al. (2021). "Status of the transient library: Transient simulation of complex integrated energy systems". In: *Modelica Conferences*, pp. 187–196.

Stüber, Moritz (2017). "Simulating a Variable-structure Model of an Electric Vehicle for Battery Life Estimation Using Modelica/Dymola and Python." In: *Proceedings of the 12th international Modelica conference*, pp. 132–031.

Tinnerholm, John, Adrian Pop, Andreas Heuermann, et al. (2021). "OpenModelica. jl: A modular and extensible Modelica compiler framework in Julia targeting ModelingToolkit. jl". In: *Modelica Conferences*, pp. 109–117.

Tinnerholm, John, Adrian Pop, Martin Sjölund, et al. (2020). "Towards an Open-Source Modelica Compiler in Julia". In: *Asian Modelica Conference 2020, Tokyo, Japan*. Linköping University Electronic Press, pp. 143–151.

Utkin, Vadim (1977). "Variable structure systems with sliding modes". In: *IEEE Transactions on Automatic control* 22.2, pp. 212–222.

Webber, Diana L and Hassan Gomaa (2004). "Modeling variability in software product lines with the variation point model". In: *Science of Computer Programming* 53.3, pp. 305–331.

Wetter, Michael et al. (2014). "Modelica buildings library". In: *Journal of Building Performance Simulation* 7.4, pp. 253–270.

Zimmer, Dirk (2010). "Equation-based modeling of variable-structure systems". PhD thesis. ETH Zurich (Switzerland).