Semantic validation of physical models using role models

Roland Samlaus and Peter Fritzson

Linköping University Post Print

N.B.: When citing this work, cite the original article.

Original Publication:
Roland Samlaus and Peter Fritzson, Semantic validation of physical models using role models, 2015, Simulation (San Diego, Calif.), (91), 4, 383-399.
http://dx.doi.org/10.1177/0037549715580174
Copyright: SAGE Publications (UK and US)
http://www.uk.sagepub.com/home.nav

Postprint available at: Linköping University Electronic Press
http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-118063
Abstract The complexity of models for the simulation of physical systems is steadily increasing. This makes the effective validation of models for different design aspects crucial. One of the many important aspects is the structural correctness and the behavior due to design parameters which are of particular concern for the modeling of wind turbines.

This article presents a design and implementation of a role-based validation framework. The framework allows for the creation of validation rules for different design aspects. This is done by role models that are used to define restrictions for an aspect by roles and rules. Multiple role models can be combined to cover all design features during model development. Restrictions on how models can interact with each other can be defined, which broadens language specific restriction capabilities. The resulting rules can then be tested on arbitrary models based on the Eclipse Modeling Framework, for which mapping between elements of the role model and elements of the validated modeling language must be provided.

In the domain of wind turbines, this approach is evaluated by application to two kinds of modeling languages (Modelica and UML2). Role models and rules have shown to be easily described with the framework’s role model language and role model definitions are successfully re-used by the definition of mappings for both kinds of modeling languages.

Keywords: Modelica, OneModelica, role models, validation, semantic constraints

1 Introduction

The development of models for simulation and assessment has become an important task in the design of new physical systems. Simulations need to be performed in order to assess the suitability of a given design in regards to desired performance and durability. Simulation results are used to demonstrate compliance to industry standards and for the certification of system components. Recurring design cycles demand an optimal structure for various aspects like cost reduction, optimal performance, or safety. In this article, the different kinds of design aspects to be taken into account are expressed by role models. Reenskøg [1] states that role models provide a way to describe the collaboration between objects. While objects in object-oriented languages have a certain state and behavior that is analyzed during development, the collaboration of objects is usually not considered. Objects can perform several roles, for example the generator of a wind turbine converts torsion from the drive train into electrical power but it also influences the vibration frequency of attached components like the nacelle and its mass. The collaboration of objects for these different aspects can be modeled with role models. Recent modeling languages that are used for the development of physical models, like Modelica [2, 3], assist engineers efficiently in creating component-based models through a domain specific syntax. By separating models into components they can easily be distributed and ultimately reused by other engineers. Free libraries (e.g., the Modelica standard library[1]) are available for commonly used components while commercial libraries providing highly detailed models can be purchased for special purposes (e.g., the power train library [4] from DLR).

[1] https://www.modelica.org/libraries/Modelica
Aside from textual representation, physical models can also be created graphically by using a Modelica connection editor or specialized graphical modeling languages like ModelicaML.

ModelicaML [5] is a UML-Modelica profile for the UML2 standard and thus builds on an industry standard that is enriched by stereotypes for a domain specific behavior. ModelicaML also allows generation of simulatable Modelica code from annotated UML models. This article uses ModelicaML as a second language since it defines physical models in a different format (UML instead of Modelica) but yet represents the models in a similar way by using component-based structures (and ultimately generates identical simulation code). However, the herein presented validation framework can potentially be used with any model that is based on the Eclipse Modeling Framework (EMF).

Models of both kinds, Modelica and ModelicaML, are connected in a similar way due to their component-based structure. Components are coupled by linking two connectors to each other. Only two connectors of the same type are permitted to be linked. Frame connectors from the Modelica standard library are used for the mechanical systems shown in the examples. While this is the only restriction for the application, incorrect specifications may result when connecting two components that do not fit together semantically. At best, the user will get an error message about the faulty model from the simulation software. For large models it can be time consuming to detect and fix such connection errors since Modelica models must first be translated into a computable form (usually C code) and the modeling error may occur in an advanced stage of the simulation. The presented framework allows the user to check for such errors during the development of physical models. Fig. 1 displays error markers and error descriptions for incorrectly connected components in the Modelica editor.

Outside of the structural requirements on the correctness of physical models, additional design aspects need to be considered during validation. Wind turbine models for example are defined by connecting different kinds of sub-components to each other. The typical wind turbine (see Fig. 2) consists of a tower that is connected to a nacelle. The nacelle contains a drive train that is connected to the generator which generates electrical power from the torque produced by the rotor. The drive train is connected to a rotor mounting one or more rotor blades (the typical design has three rotor blades).

While the experienced modeler knows the structure of wind turbines and hence may immediately recognize incorrectly connected components when they are visualized in a connection editor, the same user may have problems seeing the errors in a textual editor or when creating highly detailed components. Furthermore, when using model components from libraries, the library designers may want to restrict the way in which the model components are used in order to prevent the creation of models that would be inconstructible in real life.

An example in the area of wind turbine modeling is the existence of components of the same kind but parameterized in a way that makes them incompatible to components with differing parameter sets (this is also called a multi-point constraint in [6]). The rated power of a turbine requires that components like generators and hubs are designed for this specific power rating, e.g., 5 MW or 10 MW. Physical models of generators thus may be structurally identical but different parameters make them specific for a certain rated power class. In Modelica this may be expressed by naming conventions for the models (e.g., Generator5MW and Generator10MW) indicating special parameter sets to the
user. This convention becomes problematic when further aspects of a model design must be expressed and automatic validation is desired.

A real world example, where the validation of different parameter sets for structurally equal models is important, are load calculations for the certification of wind turbines based on different simulation set-ups. From the experience of our engineers this can be an error-prone job when simulating more than a thousand load cases for a single wind turbine design. A load case defines which environmental parameters such as wind speed and maximum wave height, are used for a simulation and how the structure should respond to the resulting loads. It can happen quite easily that components are not suitable for a certain load case which will result in redundant simulations that slow down the certification process significantly.

The proposed role model\(^2\) based validation system developed at the Fraunhofer Institute for Wind Energy and Energy System Technology (Fraunhofer IWES) allows these aspects to be expressed explicitly and models to be validated automatically. Hence, adding more semantic information to models can point modelers to incorrect design decisions, the development process can be accelerated, and the resulting designs are more likely to meet the design criteria.

The concept of role models tackles a problem with instances of language elements: they do not provide information for certain design aspects concerning their collaboration within a program. Role models define design aspects explicitly and language elements can be annotated with the roles that they perform within an aspect. Therefore, the use of language elements can be further restricted for design aspects that apply to the current program code.

In this paper we will demonstrate how role models combined with rules can help define constraints for physical models in a wide array of design aspects. The overall concept of the framework’s role model constraint language (RMCL) and role model mapping language (RMML) and how they are used to define roles and constraints for physical models is described in Sect. 2. Sect. 3 explains the syntax of the RMCL. Sect. 4 shows how elements of the RMML need to be mapped to elements of the target languages (i.e., the metamodels of physical models). An evaluation of the role model validation framework follows in Sect. 6 using both model types mentioned above. For the evaluation of Modelica models, components from our wind turbine library are used [7] which comprise a full-scale wind turbine model for aero-serve-hydro-dynamic simulation. Sect. 7 emphasizes related work that had an impact on the implementation of the validation framework. Finally, Sect. 8 gives a conclusion and outlook for future work.

1. \(^2\) (or role-constraint model, since the purpose in the framework is to define constraints on the collaboration of objects)

2 Concept of the Role Validation Framework

This section addresses the structure of the role-based validation framework. The technologies currently in use are listed and languages that have been implemented to allow the definition of role models, constraints (RMCL) and mappings (RMML) are explained. Finally, the validation mechanism is described.

2.1 Supporting Various Modeling Languages

Eclipse is used as the base technology for its ability to add custom functionality through a plug-in mechanism (see Fig. 3). The EMF provides an environment for the definition of Domain-Specific Language (DSL)s and a validation framework that is designed for use with models created with the DSLs. Language definitions (i.e., metamodels) define an abstract syntax that is detached from a concrete syntax. Hence, models created with EMF can be displayed generically with the provided tree or diagram editors. Furthermore one or more concrete syntaxes can be defined for an abstract syntax by defining custom icons for diagram editors or providing a textual representation that allows parsers to create an Abstract Syntax Tree (AST) from a textual document.

Our wind turbine models are developed with the DSL Modelica. A Modelica IDE [8] has been implemented with Xtext [9] to provide parsers which create EMF based syntax trees from Modelica code. Hence, Modelica models can be used with the role model validation framework presented herein. The second use case is UML2 models which are annotated with domain specific information provided by ModelicaML. The UML2

![Fig. 3 Technologies hierarchy of the role validation framework.](image-url)
language\(^3\) is also defined with EMF, which means that UML2 models created with this language can also be used within our framework.

2.2 The Validation Framework’s Languages

Two languages have been developed for restricting models with constraining roles:

1. The role model constraint language RMCL (Sect. 3) allows to define role models and roles for arbitrary context objects. Context objects trigger validations and need to be mapped to entities of models that are being analyzed. As a starting point context objects have ports that link to roles which are queried from the context object. Additionally, restrictions (rules) are defined with logical expressions to specify how the roles can be used in context with each other. In the examples shown, the connection of component based physical models are restricted so that the connections made are the context objects and the connected components are queried for the roles that they perform. The RMCL creates an abstract restriction model which is independent of a target model.

2. The role model mapping language RMML (Sect. 4) defines which instances of language elements (e.g., a Modelica ConnectClause or UML2 Connector inside a class diagram) can be associated with context objects defined through the RMCL. Queries for each port of the context object need to be defined for the modeled language to obtain the roles from the validated objects. The roles need to be defined as strings so that they can be associated with the previously defined roles by the validation algorithm. This is done through annotations in Modelica models or through comments in UML2 models (see Sect. 6 for example implementations).

By applying these languages it is possible to define roles and constraints for arbitrary EMF-based data structures. For each metamodel (each language being used for the development of physical systems, e.g., Modelica, UML2 or Petri nets) the mapping has to be defined once and can be re-used by other developers. As a last step, models need to be annotated with the roles that they perform.

2.3 Performing the Validation

Now that the validation framework has been set up, models can be defined and connected to each other.

The framework registers an EMF validator that validates all documents of mapped languages present in the workspace, i.e., languages for which mapping to a role model is available. The validation of edited models is automatically triggered when a model is saved or when the workspace is being built. Alternatively a validation can be triggered manually. Roles that are defined in the model elements are obtained and the constraints checked based on the role data. Error markers indicate violations of rules in creating valid models to guide developers.

2.4 Granularity of Rule Definitions

For the validation of component based models with connectors and connections, two different kinds of role definitions can be distinguished. They differ in the granularity of the defined roles that components can perform.

- The general role describes which two components can be used with each other. As an example, the role Tower for a tower component and the role Rotor for a rotor component cannot interact with each other when targeting structural validity. A nacelle component is missing which is the linking object between the structures.

- In addition to the component-wide roles, fine grained roles for single ports of a validated object are needed. This enables the definition of rules that would allow only the top frame of a Tower to be connected to a Nacelle whereas the bottom frame requires connection to a Substructure or Ground. Where only general restrictions are possible, the distinction between top and bottom connections could not be expressed. In that case, given rules would allow the connection of the Tower to Nacelle, Substructure, and Ground. In this case, a connection between the bottom frame of a Tower and the frame of a Nacelle would be valid but unrealistic.

Since the user can define which objects in his or her models should be annotated with roles and how the roles can be queried, he or she can define different types of granularity as needed. Also the use of the validation framework is not restricted to component-based models and thus provides a flexible way for the definition of restrictions for arbitrary model types.

3 Defining Role Models and Constraints

This section explains how the RMCL can be used to define validation context objects, role models, roles and constraints. The language has been created with Xtext\[^3\]
(Fig. 4 displays the metamodel) which automatically generates a text editor with syntax highlighting, syntax checks, and name based reference resolving. This supports the editing process by guiding the user through defining correct role model definitions.

3.1 Context Objects, Role Models, and Roles

Role models describe the collaboration of objects of a program for a certain design aspect like the structural validity of physical models or the suitability for specific environmental conditions. Role models define a name and contain roles that can be performed by components. Since each role has a name, qualified names (e.g., `rolemodellname:rolename`) can be used to annotate models with the role that they are performing. Rules can be defined for the roles inside a role model to restrict the collaboration between objects. In order to create role models, the first task for the role designer is to define a context object which triggers validations and which will be marked with error markers in case of rule violations. Context objects can have arbitrary ports which will be used for querying roles. For component-based models the context object could be a binary connection\(^4\) with a left and a right port, pointing to the connected components. Listing 1 displays an example from a RMCL document defining a context object with two ports.

```plaintext
context ContextObject {
  port portA
  port portB
}
```

Listing 1 Defining a context object and ports.

Now that a context object is defined and available, the second step is to define role models, roles, and rules. Role models contain a collection of roles that represent a certain aspect of validation. The examples in Sect. 6 showcase structural and power design aspects. The restriction through roles is necessary since the languages used do not limit the connection of components sufficiently. For example, it is possible to connect any Modelica components to each other as long as the connectors are of the same type. This can lead to the creation of incorrect models. Listing 2 displays an example of a role model with two roles and one rule, targeting the previously defined context object.

```plaintext
rolemodel rolemodellname {
  roles
    role RoleA "Comment for RoleA"
    role RoleB "Comment for RoleB"
  rules for ContextObject {
    (portA=RoleA implies portB=RoleB) and
    (portA=RoleB implies portB=RoleA)
    ERROR "RoleA can only connect to RoleB"
  }
}
```

Listing 2 Defining role models, roles and rules.

3.2 Rules

The third step is to define rules for context objects which restrict how the roles can collaborate. Roles can be retrieved from the context object’s port. In Listing 2 the rule says that objects performing `RoleA` can only interact with objects performing `RoleB`. This is done by a logical statement saying that if a query on the context object for `portA` retrieves the role `RoleA`,

\(^4\)in the case of Modelica, the connection is modeled as a `ConnectClause` and the validated connection object is created by a parser.
then a query for the port `portB` must retrieve the role `RoleB`. For bijectivity the statement is defined in a second statement and combined with a logical `and`. In this example no other role will be allowed to interact with `RoleA` and `RoleB`. The following logical statements are currently supported for rule definitions by the language (corresponding to propositional logic):

- **and**
  Returns `TRUE` if the left-hand side and the right-hand side evaluate to `TRUE`. Otherwise it returns `FALSE`.

- **or**
  Returns `TRUE` if the left-hand side or the right-hand side or both sides evaluate to `TRUE`. Otherwise it returns `FALSE`.

- **implies**
  Returns `TRUE` if the left-hand side of the expression evaluates to `FALSE` or the left-hand side and the right-hand side evaluate to `TRUE`. Otherwise it returns `FALSE`.

- **not**
  Negates the subsequent expression.

The statements `and` and `implies` were used in the example of Listing 2. For the support of cardinalities in rules further language elements are available. Rules can start with `for all(roleA=RoleA or roleA=RoleB)` to select a subset of context objects that are defined inside the same parent object and that are associated with the stated roles. For example in Modelica, all connections that link to a component with a certain role can be selected from a model.

The relational operators `<, <=, >, >=, ==,<>` are used to compare two cardinalities with each other. Cardinalities can either be integers or ranges (e.g., `1...3`). The function `card(roleName - portA,ports)` is used to select roles from multiple ports of a context object. The first argument of the function defines the role name that is selected. The list of arguments following the dash defines which ports of the context object are considered in the selection as described above. The number of selected roles can then be compared using relational operators to check if the cardinality is permitted.

Examples of the use of cardinalities in rules can be found in Sect. 6. Every rule must define a severity that corresponds to Eclipse’s resource error scheme (error, warning, or info). Additionally, an error message providing a description of the problem for the user must be defined. Erroneous objects are marked with the provided information to inform the user directly within editors. Additionally, Eclipse provides a separate view for displaying all errors that have occurred.

### 4 Mapping the Role Models to Target Languages

The previous section showed how role models, roles and rules can be defined with the RMCL. Since the RMCL-language elements are not bound to specific models and can be reused, mappings to target languages must be provided.

#### 4.1 Relation of Role Models and Modeling Languages

Role models refer to context objects which are the starting points for validation. Context objects need to be mapped to elements of the modeling language. This is a one to one relation. The context objects define ports which are used for the definition of rules. The ports point to roles which have to be queried from instances of the modeling language. Hence, starting with the context object, the user needs to define a query that collects all role definitions from model instances. Queries are defined within role mapping documents and are used to collect role definitions from model instances (see Section 4.3).

When a context object is validated, the role definitions are collected and grouped so that all roles belonging to a role model can be accessed during validation. The role models that refer to the context object are then selected and the rules that are defined for a role model are validated with the role definitions that have been found. Fig. 5 displays the mapping from context objects and roles to language instances.

#### 4.2 The Role Model Mapping Language

For each target language, a mapping must be provided. The metamodel of RMML shown in Fig. 6. Listing 3 displays basic statements of an RMML document.
In the first line the target language is referenced by its namespace URI. The second statement indicates which role model is being mapped. The statements are used to resolve references to elements of the target language and the RMCL to immediately provide the user with information whether referenced objects exist and queries can be evaluated. Additionally, proposals for structural features of the context objects that are being mapped can be calculated by introspection, assisting the user during the mapping definition.

In Listing 4 the context object ContextObject from the role model example in Sect. 3 is mapped to a class definition with the name ExampleClass from the referenced language.

Instances implementing the class ExampleClass will trigger validation as described earlier. The query definitions of each port from the mapping document are used to receive the role definitions from the context object. In the example, the LeftQuery and RightQuery are used to receive the role definitions for the left and right ports from the ExampleClass instance. The queries ultimately need to retrieve an object of type String that contain one or more role declarations.

4.3 Querying Roles from Context Objects

A custom language has been created for the definition of queries. There are several existing languages that could be used for this purpose since they allow querying of EMF-based models [10,11,12], but they have some drawbacks in handling large models. The present language therefore tries to simplify the querying of large models. The user defines how the roles can be received from model instances. This is based on the model’s structure as defined by it’s metamodel. Thereby, the models are analysed regarding their static semantics as defined by the roles. If an error occurs while querying a model, an empty set of roles is returned. Possible errors include a query receiving an object that is not of type String, or a referenced object that does not exist and thus would lead to a potential Null pointer access. Another simplification is to not differentiate between single objects and collections of objects. Queries can be defined in the same way and there is no need for special set operations. However, selections are still possible like those for objects of a certain type.

4.4 Query Language Concepts

Listing 5 shows the RMML concepts that are currently available to the user. Line 1 displays a simple query. A structural feature is selected from an object, for example from the context object at the beginning of a query. A structural feature in EMF is an attribute, a reference, or an operation of a modeled class.

Listing 5 Query concepts of the mapping language (RMML).

```java
Listing 3  Defining the target model and the mapped role model.
Listing 4  Mapping connections and ports to the target language.
Listing 5  Query concepts of the mapping language (RMML).
```
The type of the object may not be statically known if the object has been retrieved from a reference with an abstract type definition. In Modelica, for example, equations in an equation section can be of type ConnectClause or EquationExpression (see Fig. 8). Hence, the structural features cannot be statically derived. To avoid this, the user can define a cast to a specific type as demonstrated in line 2 if he knows that only one type will appear.

In line 3 a function is called that needs a return value. Two types of functions exist in the RMML. One type is modeled using EMF and is generated or hand-written in the generator's target language, e.g., Java. The other type of functions is defined directly with the RMML language and is used to gather recurring operations, like common queries. The currently queried object will then be provided as an input parameter for the functions.

In lines 4-6 a case distinction is used to select the correct query for the current object by type-checking the object. This is also used when a list of objects of different kind is being processed. In the example QUERY_A is selected if the investigated object is of type TypeA. The third entry states that QUERY_C is selected if the current object converted to a String equals the given value.

The last two lines handle lists of objects. While in line 7 all elements of a type are selected, in line 8 a single object at a specific index in the list is selected.

5 The Validation Algorithm

It is important to understand in regards to the validation algorithm that the role model validation framework keeps track of all RMCL and RMML documents inside the workspace. As soon as a document is added, changed, or removed, the information concerning the affected resource is updated in the registry. As a result, up-to-date information about all mappings, roles, and context object definitions are available during the validation of model instances.

Fig. 7 shows the objects that are involved in the validation process. All documents for mapping and role definition which are present in the workspace are taken into account during validation. Since rules are interpreted new documents can be added dynamically. If standard rules and mappings are expected to be provided in the future, they can be added via an extension point. However, this feature is currently not implemented. Two separate tasks occur during validation: The logical validator uses a query engine to receive role definitions (defined as text) from model instances. The logical validator uses the provided roles to check if the model element under investigation violates rules. The validation result is then passed back to the role model validator which will mark model elements with an error marker if a violation has been detected. The validation algorithm is described in detail below:

1. The EMF validation framework triggers the role model validation framework's validator (role model validator in Fig. 7) upon model change, workspace build or manual activation. The validation starts with the root element of a document and continues with all sub-elements until all model elements inside the document have been validated.
2. Initially all previously added error markers of the current element are removed.
3. The validator checks if a mapping from the model element's class to a context object exists in the registry. If not, no validation will be performed for the object at hand and the algorithm continues with step 1.
4. The role model validator now associates the model element with the context object definition. For each port definition of the context object, the queries that are defined inside the role model mapping documents are performed by the query engine on the current model element. If role definitions can be found by the query engine, they are kept in a list that is associated with the port and passed over to the logical validator.
5. The role model constraint documents contained in the registry provide the rule definitions for the context object of the current model element. For each rule, the roles of the current model element are filtered in order to select only valid role names. For example, in Listing 2 the roles with the qualified names RoleModelName.RoleA and RoleModelName.RoleB are selected.
6. Valid roles as well as rules from the role model constraint documents are passed on to the logical validator. The roles are verified by the logical validator according to the logical statements described in Section 3.2. The example shows that Listing 2 checks if the role RoleA is defined for portA. In this case, the RoleB must be defined for portB and vice versa.
7. The validation results are handed over to the role model validator. If the validation of a rule returns a negative result, the current model element is marked with an error marker that is provided by the Eclipse framework. The error message from the rule definition is used to describe the type of error.
6 Evaluation

Within the domain of wind turbines many design aspects need to be considered when highly detailed models are created (e.g., the need for sealed components for offshore wind turbines). As the ability of collaboration between components grows, it becomes more important to check the compatibility of these systems in order to assist the user in tracking all design aspects.

This section defines role models and constraints for two kinds of modeling languages: Modelica and UML2. The focus lies on two different design aspects. One that targets the overall validity of the model structure and one validating that composite wind turbine models are designed for the same power rating.

6.1 Role Models for Structural Validity and Rated Power Design Aspects

As a first step, role models, roles, and rules need to be defined for the design aspects in question. Listing 6 displays three constraints enforcing correct connection of tower, nacelle, rotor blade, and hub connectors as well as one constraint that defines the need for a wind turbine to have a connection between a nacelle and a tower. Roles are defined for the top connector of a tower, the connector of a nacelle linking to a tower, a rotor blade connector, a connector of a hub linking to a rotor blade, and a wind turbine. The first two constraints define the following: only models with the role Nacelle_To_Tower can be connected to models performing the role Tower_Top; there must be exactly one connection between these roles. The third constraint defines that only 1 to 3 rotor blades can be connected to a hub.

![Role Model Validation Framework](image)

**Listing 6** Roles for structural validity.

These restrictions represent typical instances in wind turbine design. It is generally understood, that a connection between a tower and a rotor is unrealistic. However, in large textual Modelica files with many connect equations, this kind of error might be hard to identify. In real life application the use of one, two, or three rotor blades is common for fast rotating wind turbines. Mounting additional rotor blades could lead to structurally unstable behavior.

Model components must fulfill requirements regarding reliability in order to get certified. Guidelines [14, 15] define load scenarios that need to be simulated and the specified parameters vary with the power rating of the wind turbine design. During the creation of simulation models it is easily possible to use incorrect components which would result in time consuming, erroneous, and unnecessary simulations being performed repeatedly to produce valid results. Listing 7 contains two simple constraints that target the rated power levels of 5 and 10 mega watt wind turbines. For different power levels the models usually only differ in magnitude of parameters. The drive train of a 10 mega watt turbine must bear more torque than the drive train of a 5 mega watt turbine which results in larger dimensions and a higher mass. When components are connected that are annotated with constraints, the power rating of each component can automatically be verified for compatibility.
This will prevent a modeled drive train with a rated power of 5 MW to be used in 10 MW wind turbines.

```
roles
role R_P_10MW "10 MW components"
role R_P_5MW "5 MW components"

rules for BinaryConnection{
  (left = R_P_5MW implies right = R_P_5MW) and
  (right = R_P_5MW implies left = R_P_5MW)
  ERROR "Only 5 MW rated power components can be connected to each other."

  (left = R_P_10MW implies right = R_P_10MW) and
  (right = R_P_10MW implies left = R_P_10MW)
  ERROR "Only 10 MW rated power components can be connected to each other."
}
```

Listing 7 Roles for rated power design.

In such a scenario the torque generated by the rotor would exceed the maximum capacity of the drive train in strong wind conditions. In real world applications the components could not be connected because the flange of a 10 MW rotor would be too large to be mounted to a 5 MW drive train. In modelling, however, the difference in the components is most likely modeled by different parameter sets. Hence, the models of 5MW and 10MW drive trains are equal except for the parameters, which makes it difficult for non-experts to choose suitable components when creating new wind turbine models.

The combination of role constraints can be useful since structural rules and rated power related rules can be used side by side. This is even more useful when another modelling task is taken into account which concerns the location of the designed wind turbine. If wind turbines are located offshore special components, such as sealed nacelles, have to be used that can withstand the rough environmental conditions at sea. Components can be annotated with roles for offshore or onshore use which would allow the user to check whether a wind turbine model contains only components that are designed for 10 mega watt rated wind power and is suitable for offshore weather conditions. Hence, further rules can assist the user in creating correct models for numerous kinds of design aspects.

6.2 Mapping the Role Models to Modelica

Now that the roles and constraints are defined using the RMCL, mapping between the elements of the RMCL and the Modelica language needs to be defined with the RMML in order to validate the Modelica wind turbine models. We will show mappings to BinaryConnection roles for Modelica and mappings to MandatoryConnection roles for UML2. The Modelica language is complex and therefore we will discuss a simplified language (see Fig. 8) at this point. The complete language definition and all mapping definitions can be downloaded from our website\(^6\). ModelicaSourceFile represents the root of a Modelica document and is the entry point for the parser creating the parse tree. A Modelica document can contain several classes. Each class can have elements such as import statements, subclasses, and many more, but only EquationSections containing ConnectClauses are of interest in our example and therefore the other types of elements are omitted at this point. A ConnectClause has two references (left and right) to Components that are defined inside classes and which are of type Connector. ConnectClauses are defined as context objects in the RMML document and will trigger validation. The roles are retrieved by following the references left and right to the connected components that are of type Connector. Now the annotations of the components are analyzed for Strings that define roles. Annotations can be quite complex as well, so be aware that the reference text is a shortened expression which actually retrieves the role definitions as a String from the Annotation.

---

\(^6\) [http://www.onewind.de/RMVF.html](http://www.onewind.de/RMVF.html)
Listing 8 displays the Mapping for the simplified Modelica language.

target "http://www.osewind.de/language/Modelica"
maps "platform:/resource/role/src/Contexts.role"
costextmapping BinaryConnection = ConnectClause {
    left = ConnectClause.LeftQuery,
    right = ConnectClause.RightQuery
}  
RoleQuery ConnectClause.LeftQuery {
    case ConnectClause :
        left.selectAnnotations().text
}  
RoleQuery ConnectClause.RightQuery {
    case ConnectClause :
        right.selectAnnotations().text
}  
function Annotation selectAnnotations(Component) {
    return type.elements.all(Annotation)
}

Listing 8 Mapping for Modelica.

The RMML document starts with the declaration of the target language via its URI (Uniform Resource Identifier). In the second line the document containing the roles is referenced. Then a context mapping is defined stating that a BinaryConnection in the RMML document corresponds to objects of type ConnectClause.

For the ports left and right queries are provided which retrieve role definitions from the context objects. The role queries start with a case statement checking whether the object provided for query is of type ConnectClause. If this is true the left and likewise the right references to a component are queried and the function selectAnnotations is called which obtains the retrieved Component as an input. In this function the elements of the component’s type are taken, the Annotations are selected and then returned. Then the text reference of each Annotation - which is a String - is returned.

Fig. 1 from the introduction displays a screenshot of the Modelica editor showing parts of the specification for a wind turbine model.

The last connect equation attaching frame number 4 of the hub to the frame of blade number 4 in line 18 is marked with an error since it exceeds the limit of a maximum of 3 connections. The subsequent connect equation violates the constraint which says that only one connection between a tower and a nacelle can exist. Finally the last connect statement violates the restriction that the tower top frame can only be connected to the appropriate nacelle frame.

6.3 Mapping the Role Models to UML2

The same rules used above for the Modelica models shall now be re-used with wind turbine models created with ModelicaML. Fig. 9 displays an extract of the UML2 language definition. Listing 9 starts with a reference to the UML2 language definition and the role definition file that contains the objects to be mapped.

target "http://www.eclipse.org/uoml/4.0.0/UML"
maps "platform:/resource/role/src/Contexts.role"
costextmapping MandatoryConnection = Class {
    mandatory = ClassRoleQuery,
    left = ClassLeftPortQuery,
    right = ClassRightPortQuery
}  
RoleQuery ClassRoleQuery {
    case Class : ownedComment.body
}  
RoleQuery ClassLeftPortQuery {
    case Class : ownedConnector.end[0].role.ownedComment.body
}  
RoleQuery ClassRightPortQuery {
    case Class : ownedConnector.end[1].role.ownedComment.body
}  

Listing 9 Mapping for UML2.

The MandatoryConnection context object from the role definition is mapped to UML2 Class instances. For the definition of roles inside UML2 models comments are used that can be attached to UML2 models. The ClassRoleQuery is defined for the mandatory role which generates the body of the ownedComment from the Class. The left and right queries generate the UML2 connectors, which represent ports of classes that can be connected to other ports from the investigated objects. They take the left and the right end of the connection respectively (which corresponds to the first and second entry in the list of connectorEnds). The roles are se-
Now that the mapping is defined, UML2 models can be annotated and checked against the rules previously defined. Fig. 10 displays a wind turbine model in the ModelicaML diagram editor. The faulty connections are marked with an error icon. Additionally, the Eclipse problems view indicates which errors have occurred. Compared to tree and text based editors, the diagram editor has the advantage that faulty connections can easily be recognized visually without the need for additional validation.

However, this is not the case for errors regarding the permitted amount of connections since the user could possibly create arbitrary connections between the hub and rotor blades. The same holds true for power rating constraints where the types of components which are being connected to each other are of importance. Any errors in these instances cannot be recognized immediately by using diagram editors and demands for automatic validation.

In addition, rules were defined for the definition of operating conditions with low wind speeds, which are common for in-land turbines, and for operating conditions with high wind speeds, which are common for offshore turbines. Fig. 11 displays a simple wind turbine model with annotated components in the Modelica editor.

The two components gearbox and rotor in the wind turbine model HSWindTurbine can be connected to each other since the connection does not conflict with any constraints. The second connect statement however, is not allowed since the slowgearbox component is designed for in-country use with slow average wind speeds while the rotor is designed for high wind speed environments. The third connect statement violates the role constraints from Listing 7.

6.4 Results

Once a mapping has been defined roles can be reused with models of the mapped languages. The complexity of the language will determine the effort necessary for mapping. We were able to show that the framework can be used with different kinds of EMF-based models and error markers are created to display the inadequate use of components. The rules defined for structural validity assist our students in learning how to create wind turbine models with Modelica. Further rules need to be defined in order to restrict all important scenarios that exist in wind turbine modelling.

Other kinds of role constraints help to assist the user in selecting the correct components for a certain design scenario. The combination of different role models helps to build a strong validation framework that checks the model against various aspects. Hence, the selection of components from large libraries as well as the devel-
opment of new models can be assisted especially when many variations of the same component already exist.

7 Related Work

The idea of role models that express the collaboration between objects was initially described by Trygve Reenskaug et al. [1]. In the proposed Object Oriented Role Analysis and Modeling method, roles are used to describe the collaboration for certain design aspects and views are provided for presenting the objects collaborating for an aspect. Objects can play multiple roles where each role is qualified by aspects.

Dirk Riehle and Thomas Gross demonstrated [16] that role models can be used for the design of software frameworks. They added semantics to the framework classes on how they can be used in collaboration by defining rules for the roles the classes play. As an example, they showed how the hierarchy for figures in a visualization framework can be defined by applying rules for parent-child relations to framework classes.

Role models were successfully used for generic refactorings [17]. The framework allows to define refactorings in a generic way e.g., to rename objects or to move extract objects into a new container object. Similar to this work, roles and rules for the refactoring are defined with custom language. The definitions need to be mapped to the metamodel of each target language that can be used with the refactorings. The applicability has been demonstrated with several DSLs.

[13] presents a first approach for semantic validation of Modelica models with OCL. OCL as a common language for static semantic validation of EMF models allows the definition of constraints that can be automatically validated by the EMF validation framework. Constraints are also defined for context objects and the validation is triggered in the same way as validation with the proposed role model validation framework. One drawback is that constraints require re-definition for each target language with OCL. When one rule needs to be changed, every OCL-constraint requires modification accordingly. Also a meta model changes should be reflected in each constraint for the modified language since the constraints are defined directly based on the meta model. These problems have been avoided by the role model validation framework through separation of constraint definition and language mapping. Finally, it was mentioned that the performance of interpreting OCL constraints on large data sets has a negative performance compared to Java code.

When modeling physical systems, certain issues arise for component-based software systems and frameworks [18, 19]. Components provide ports that can be connected to each other in order to interact. From the syntactic point of view, components can be connected in any way as long as the ports match each other (in [20] arbitrary class members can be defined as ports for composition). [6] more generally describes how framework instantiation languages should be defined to prevent users from misuse.

The use of model driven technologies and applications in simulation modelling are addressed in [21] and [22]. Different techniques like Model-based Engineering (MBE), Model Driven Software Development (MDSD) and model transformation languages are explained and it is demonstrated how these technologies can be used to create models with DEVS formalism.

Since Eclipse is used as the framework for the products created at Fraunhofer IWES, the Meta Object Facility (MOF) implementation Eclipse Modeling Framework (EMF) is used for the definition of modeling languages. For the definition of wind turbine metamodels and models we use editors available with EMF and textual editors that are provided by and generated with the metamodeling tool Xtext. This includes the definition of a grammar for the textual language Modelica [8].

8 Conclusion and Future Work

The presented validation framework makes it possible to create roles for physical components with the dedicated textual language RMCL. Constraints can be defined for restriction of how the components are connected with each other. By using the framework’s mapping language RMML, arbitrary domain specific languages can be used for the validation of these restrictions. The validation framework was evaluated with two types of modeling languages used at Fraunhofer IWES. One language is UML2 refined with Modelica stereotypes. The other one is Modelica, a general purpose language in the field of physical modeling. Models defined with the two languages were successfully restricted for two design aspects: Some rules for structural validity and suitability for different rated power designs were applied.

Although this paper concentrates on the validation of component-oriented models by checking the connections between components, the framework could be used in other contexts. Instead of connections, references from one model element to another could be validated. The use of components can be restricted for object-oriented models, which corresponds to the approach presented in [16].

\footnote{http://www.eclipse.org/Xtext/}
In the future, the RMCL may be extended to support extension of roles by other roles. Thus, general restrictions could affect roles dedicated for special purposes (e.g., a role offshore tower could extend the regular tower role). This would reduce the effort of defining restrictions and the number of roles that need to be defined inside the models. Although the available restriction types are sufficient for the investigated component-oriented models, more expressions may be needed for a generalization.

A future topic of investigation is the change of roles depending on the state of the context object. When designing offshore wind turbines, the water depth plays a crucial role in the selection of the substructure [23]. For shallow water (up to about 30 meters depth) regular monopile structures can be used whereas in deep-water environments (60 meter depth and more) only floating structures are applicable.

For now only a small number of role models and constraints have been implemented which can help beginners to create correct wind turbine models. In the future, it would be beneficial to implement more rules for structural validity and to find other collaboration aspects where the framework can also aid experienced developers. It would be helpful for users to generate rules for the Modelica Standard Library (MSL). This would enable constraints to support a large number of users since the library is widely used. For large models with many constraints the performance of the role model validation framework needs to be investigated.

Acknowledgment

This work is financially supported by the Federal Ministry for the Environment, Nature Conservation and Nuclear Safety based on a decision of the Parliament of the Federal Republic of Germany and by the ELLIIT project, the Swedish Strategic Research Foundation in the EDOp projects, and Vinnova in the RTSIM and ITEA2 MODRIO projects.

References