ENCAPSULATION IN OBJECT-ORIENTED MODELING FOR MECHANICAL SYSTEMS SIMULATION – COMPARISON OF MODELICA AND BEAST

Alexander Siemers[1], Peter Fritzson[2], Dag Fritzson[1]
[1]SKF Engineering & Research Centre, Gothenburg, Sweden, [2]Linköping University, Sweden

Corresponding author: Alexander Siemers, SKF Engineering & Research Centre
Technology & Integration Department
SKF MDC, RKs-2
SE-415 50, Gothenburg, Sweden
alexander.siemers@skf.com

Abstract.

In systems engineering and object-oriented design, encapsulation is a key concept to handle complexity. Interfaces are defined for the external interaction of a component, whereas internal details are hidden. Complex systems such as cars or airplanes consist of many components, which in turn consist of many components – hierarchically through many levels. Therefore composition is built into modeling languages such as Modelica. External interfaces must be defined for external interaction, whereas internal components cannot be accessed if they are not available through these interfaces.

However, in 3D mechanical systems modeling and design, it is natural to be able to connect components whose surfaces are externally available. For example, the motor belonging to a car can be externally accessible in the 3D view, even though it can be regarded as an internal component of the car.

In this paper we compare two modeling approaches, one is Modelica used for systems engineering as well as modeling of multi-body systems and the other one is BEAST which is a specialized multi-body systems tool with good support for contact modeling. The current Modelica approach requires strict interfaces with encapsulation of internal components, whereas BEAST allows connections to internal components which are visible in a 3D view, which is often natural from the 3D mechanical systems point of view. For 3D mechanical systems, Modelica might be too strict, whereas BEAST might be too forgiving. Two different solutions are presented and discussed (in the form of possible Modelica extensions) to combine the advantages of both approaches.

1 Introduction

Multibody systems are used to model mechanical systems in which several bodies (components in mechanical systems) interact with each other.

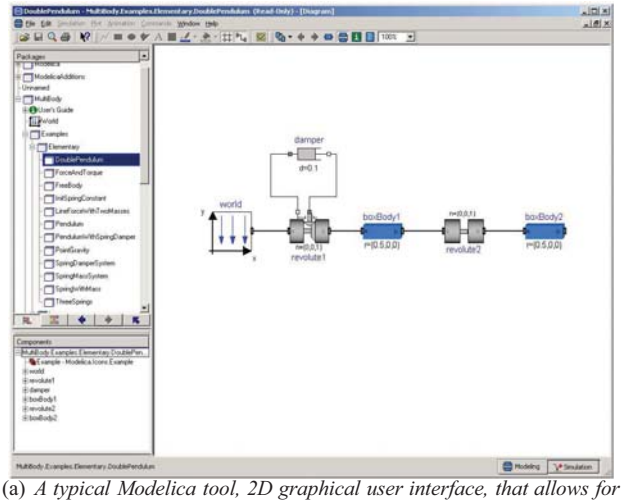
Object-oriented modeling has been applied to different application areas within the domain of mechanical system simulations. However, what are the right abstractions to represent real-world objects (bodies) in the computer? To address such problems, object-oriented programming [1] and modeling has been invented and applied to many different application areas, also within the domain of mechanical system simulations. Fritzson and Fritzson [14], for instance, apply object-oriented concepts to modeling and simulation of machine elements applications. Moreover, in this context the equation-based object oriented language `ObjectMath` [22] is also introduced for modeling and simulation in general. Another equation-based object-oriented language for multi-physics modeling is `Dymola` [17]. Both of these are ancestors of Modelica [9] [23]. Modelica's and Dymola's features allow for domain specific class libraries including multi-body systems [17] [20] [19].

More specific examples of object-oriented modeling for mechanical systems are, for instance, rolling bearings [18] and robot systems [15]. Object-oriented concepts have not only been applied to multi-body systems but also used within other areas for mechanical system simulation, e.g., finite-element analysis. Cross, et.al. [16], for instance, investigate the advantages and disadvantages of object oriented concepts for finite-element analysis. They argue that object oriented modeling has many advantages but is not useful for small problems - only for larger problems that can be divided into smaller sub-problems.

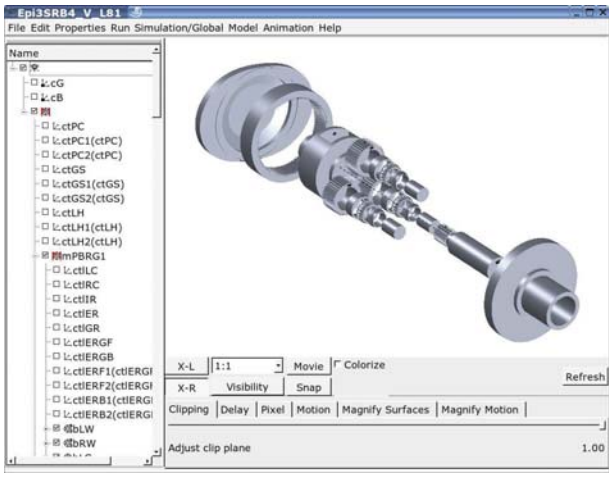
Not only are object-oriented principles applicable to different application areas but they can also be applied in different ways within the same modeling domain.

1.1 Modeling and simulation environments

The analysis and comparison presented in this paper have been conducted with Modelica [9] [23], a general equation-based object-oriented modeling language for multiple application domains, and BEAST [11] [12], a tool for object-oriented modeling and simulation of multibody systems, with special support for contact problems of rolling bearings and rolling bearing related applications.



(a) *A typical Modelica tool, 2D graphical user interface, that allows for class design and model composition.*



(b) *BEASTs graphical simulation pre-processor allows for a three dimensional model view.*

Figure 1: *BEAST and Modelica tools have graphical user interface support.*

Both the Modelica simulation environments and the BEAST simulation environment allow for graphical model composition, see Figure 1. The different characteristics of the modeling approaches are reflected in the graphical modeling environments. Pure mechanical system simulation environments, e.g. BEAST, typically support 3D visual representations of the simulation model. Another example of a mechanical system tool supporting 3D model editing is MSC.Adamas. On the other hand, most GUI s for systems engineering and multi-engineering environments, including Modelica-based tools, primarily use 2D connection diagrams for graphical modeling. However, in several cases (e.g. the Modelica MultiBody library) 3D visualizations/animations can be generated from such models. Also, some ongoing work introduces 3D GUIs for model editing, e.g. in the Simantics project [25]. There are also multi-physics environments such as FEMLAB, primarily for FEM/PDE problems, which has a 3D modeling GUI, but are oriented towards shaping and connecting just a few components – not a large number of components (hieachically at many levels) as in systems engineering.

1.2 Structuring of this paper

Section 2 introduces basic object-oriented concepts, with applications to multi-body systems. Section 3 presents Modelica-based tools and the BEAST tool, and how these tools are applied for multi-body system modeling. Section 4 discusses the key concept of encapsulation and contacts in the context of multi-body system modeling, with special reference to the Modelica and BEAST approaches. Section 5 compares model libraries and model composition in Modelica and BEAST; and finally Section 6 presents the conclusions.

2 Object-Oriented concepts applied to multibody-systems

A multibody system model typically contains, but is not limited to, the following objects:

- An environment object that defines the boundary conditions for a mechanical system, e.g., gravity and ex-

ternal forces.

- Self-contained objects such as physical objects (e.g., bodies) and abstract object (e.g., coordinate systems).
- Connection objects that define contact interactions between bodies as well as other links between self-contained objects.

Different classes of objects appear at different levels of abstraction in the class hierarchy. For example, a *body* class is the generalized form (a parent class) of *flexible-body* and *rigid-body*.

Objects are instances of such classes. They are typically organized in hierarchical *whole-part* or *parent-child* relations that build the object hierarchy of the multibody system, e.g., rings, rolling-elements, and cage are parts of a rolling-bearing (the whole), see Figure 2.

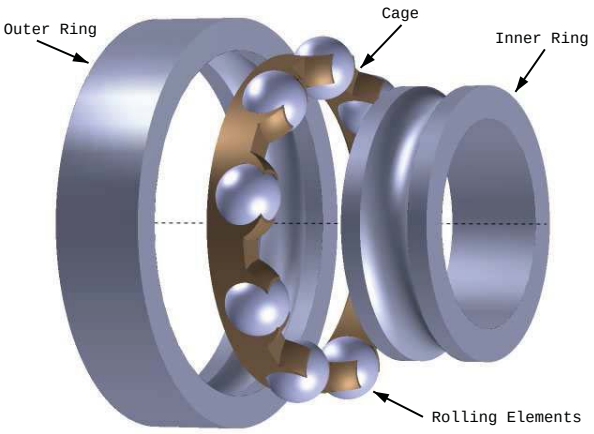


Figure 2: *A ball bearing model consisting of two rings, several balls, and a cage.*

2.1 Associations in object-oriented models

Several relations are defined between the objects and classes in an object-oriented model. An association is a relation between two objects that does not imply a hierarchical relationship between them. Objects simply know about each other.

Aggregation is a strong form of association. It describes whole-part or containment relationships. It is common to distinguish between aggregation and composition. Composition is a strong form of aggregation where the aggregate is responsible for instantiation of its components. This is not the case for an aggregation. Components in a composition can therefore not exist without the composite. The gearwheel in Figure 3 is responsible for instantiating its surface objects, i.e., there exists a composition relation between the gearwheel and its surfaces. The gearwheels on the other hand are created independently from the gearbox and added to it afterwards, which is an aggregation relation.

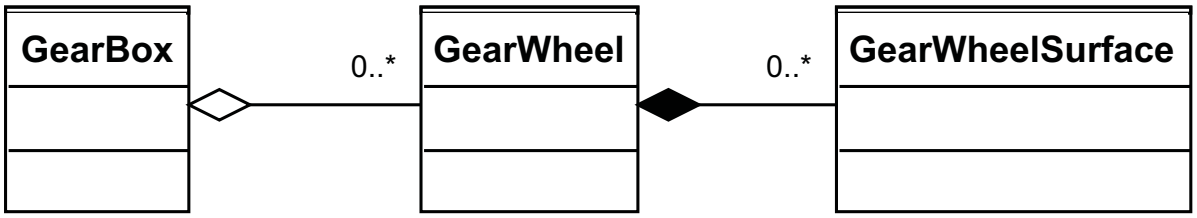


Figure 3: *Aggregation is used to assemble the gearbox from different gearwheels. Composition describes the strong relationship between a gearwheel and its surfaces, e.g., teeth.*

3 Tools for modeling mechanical system simulations

Two approaches/tools for mechanical system modeling are compared in this paper. One (BEAST) is used exclusively for mechanical system modeling, i.e., multibody system modeling. The other (Modelica) has a broader application area, i.e., multi-physics system modeling including multibody system modeling as a special case. The following main observations have been made:

- The modeling process is often very domain dependent. A general target domain of the simulation tool also requires a general modeling process. For example, general multi-physics modeling requires a high level of

flexibility, i.e., class and interface design, while pure mechanical modeling can utilize predefined classes of objects.

- Composition hierarchies can be created in several ways. One method of creating object hierarchies is object-oriented composition where the part objects are declared as belonging to a containing object. More flexible hierarchy composition can be achieved at runtime, e.g., adding or removing child objects on demand using flexible data structures such as linked lists to represent the containment relation.
- Modeling of contacts between mechanical parts requires accessible interfaces. Contacts in multibody systems are possible between any two bodies in the model that do not have any intervening object or boundary that prevents physical contact. For example, if the surface of a body is exposed to the outside world it can in principle be connected to any outside body, whereas a body which is hidden within a physical container (e.g. a box or a sphere) cannot be connected to outside bodies. If the simple solution of making all interfaces globally accessible is used, additional (geometrical) checking will be needed to prevent unphysical impossible contacts.
- Graphical user interfaces are often domain dependent. For example, (regarding 3D mechanical system modeling) mechanical systems are three dimensional by nature. In order to fully understand and verify such systems behaviour a user interface with support for three dimensional visual representations of the complete system is desirable. vs. 2D)

These issues are discussed in the following section, which also gives a short introduction to the BEAST and Modelica modeling approaches.

3.1 Modelica and BEAST/BML

`BEAST` [11] [12] is a toolbox for modeling, simulation, and analysis of detailed contact problems in bearing-related multibody systems. Besides the simulation kernel and the pre- and post-processors it includes an object-oriented C++ library for multibody system modeling. Model composition is performed with a graphical tool that allows the user to instantiate, connect, and combine models from a pre-defined BEAST library of basic models. To represent and store a user-defined model BEAST uses a special purpose language representation with a strict notation and grammar. The BEAST modeling language (BML) is a domain-specific language designed for modeling of multibody systems in order to simulate their behavior. The BEAST simulation kernel parses the BML file and creates the necessary class instances that represent the simulation model at run-time.

The BEAST library contains basic classes for modeling of complete multibody systems, including *bodies*, different classes for *connections* between bodies, and external boundary conditions. A BEAST model is a collection of interdependent instances of any of these classes. These instances are the *model components*. Moreover, there exist a hierarchical composition relation between the model components called the model hierarchy.

Additionally some other important concepts are defined for BEAST models:

- Strict naming convention, including namespaces.
- Composition hierarchies by sub-modeling. Several model components can be gathered together into a sub-model to improve the overall structure of the model, i.e., an additional hierarchical level is introduced. Several sub-model levels can exists in a BEAST model.
- Modeling and simulation of detailed contacts between bodies.

`Modelica` [9] is an object-oriented equation-based language for modeling of multi-physics systems. Modelica uses the concept of object-orientation to bring structure into large physical systems. Some of Modelica's main advantages are:

- A complete and standardized object-oriented modeling language.
- Acausal equation modeling.
- Connection associations for interaction modeling.
- A standardized graphical notation is part of the language.
- Availability of graphical model editors.

Several Modelica environments exist which support graphical and textual model design and include a simulation kernel, e.g., MathModelica [24], Dymola [10] (the Dynamic Modeling Laboratory, not the previous mentioned Dymola language), and OpenModelica [13].

4 Contact modeling and encapsulation for 3D mechanics

When dealing with 3D contact mechanics modeling there are situations where contacts occur between two parts that are not at the same hierarchical composition level. Contacts between bodies are possible if their surfaces are accessible to each other, even if the bodies are embedded in other objects from a logical composition point of view. From a contact 3D modeling point of view all bodies or their surfaces, which are exposed to the outside environment, must be accessible. Contacts in BEAST therefore have access to any child component at any lower level in the model hierarchy. Model designers (BEAST users) create the contacts that are of interest for a particular system analysis and are responsible to avoid the creation of contacts between bodies whose surfaces are not physically accessible. For instance, contacts between two physically contacting gear-wheels in a gearbox model can be neglected (or replaced by simpler joints) if they are not of interest for the current system analysis. This is typically done to improve simulation performance. On the other hand, contacts between two gears that are physically not in contact could be created by mistake in BEAST. Modelica on the other hand uses a more clean and strict approach in deep connection modeling. Connections through multiple sub-levels in the model hierarchy require the connector to be propagated through all sub-levels.

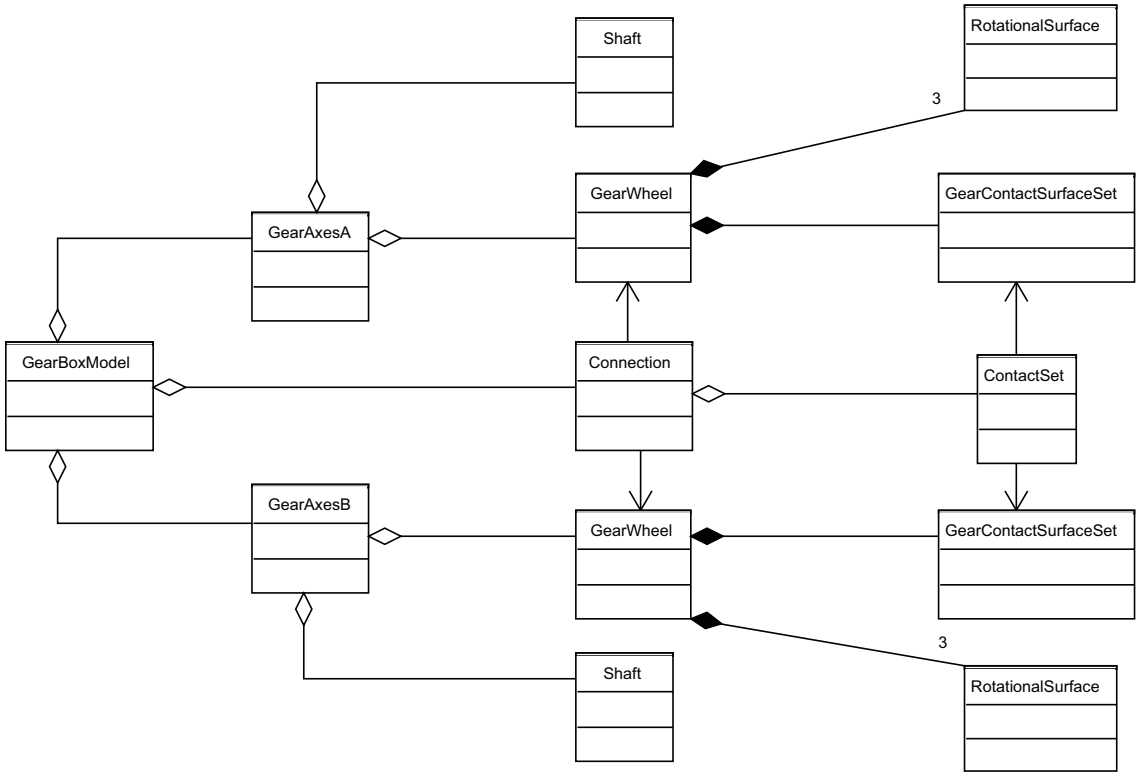


Figure 4: *Simplified BEAST gearbox model with contacts between the gearwheel surfaces. Multiple contact-surfaces and corresponding contacts are modeled as set-objects.*

As an example a BEAST model of a gearbox is shown in Figure 4. The gearbox contains two gearwheel-shafts, where a gearwheel-shaft is the combination of a shaft and a gearwheel. *Connections* are defined between the gearwheels of the two gearwheel-shafts. Contacts (*ContactSet*) are defined between the contact-surfaces (*GearContactSurfaceSet*) of the two gearwheels. The connection object that contains the contacts is placed directly in the common parent node, here the *GearBoxModel*. The BEAST model assembly is based on different factory design patterns. Any part of the model can be directly referenced. Here is shown an outline of the BML code for a BEAST gearbox model:

```
/* The Gearbox top model */
model type=SModel
      name=mGearBox

  /* The first gearwheel shaft */
  model type=SModel
        name=mGearWheelShaftA

    body type=Disk name=bShaft
      /* Shaft data here */
    end body;
```

```
        body type=GearWheel
             name=bGearWheel
          /* BEAST uses certain naming     */
          /* conventions for surfaces      */
          /* There are always two contact- */
          /* surface sets per gearwheel   */
          surface_set type=InvoluteSegSet
                      name=sFF
          end surface_set;
          surface_set type=InvoluteSegSet
                      name=sBF
          end surface_set;
        end body;
      end model;

      /* The second gearwheel shaft */
      model type=SModel
            name=mGearWheelShaftB

        body type=Disk name=bShaft
/* Shaft data here */
        end body;

        body type=GearWheel
             name=bGearWheel
          surface_set type=InvoluteSegSet
                      name=sFF
          end surface_set;
          surface_set type=InvoluteSegSet
                      name=sBF
          end surface_set;
        end body;

      end model;

      connection type=GearGearConnection
                 from=mGearWheelShaftA‘bGearWheel
                 to=mGearWheelShaftB‘bGearWheel
          /* There are always two contacts */
          /* per gearwheel connection  */
        contact_set type=ConnectBaseSegVSet
                    from=sFF
                    to=sFF
        end contact_set;
        contact_set type=ConnectBaseSegVSet
                    from=sBF
                    to=sBF
        end contact_set;
      end connection;
    end model;
```

One can see that the gearwheel is directly referenced in the connection objects, i.e., *from=mGearWheelShaftA‘bGearWheel to=mGearWheelShaftB‘bGearWheel*, and the contacts are defined between the gearwheel surfaces, called *segments* in BEAST.

A similar, very simplified, model in Modelica could look like this:

```
connector GearSurfaceSet
  // Any code here
end GearSurfaceSet;

model GearWheel
  // Gear wheel contact surfaces
  GearSurfaceSet con_surfs;
```

```
end GearWheel;

model GearWheelShaft
  GearWheel gearWheel;
  Shaft shaft;
  // An additional GearSurfaceSet
  // instance is used on the
  // GearWheelShaft level
  // for gearwheel contact-surface
  // access
  GearSurfaceSet gear_con_surfs;
equation
  // We connect the surfaces to make
  // them accessable from outside
  connect( gear_con_surfs,
           gearWheel.con_surfs );
end GearWheelShaft;

model GearBox
  GearWheelShaft shaftA;
  GearWheelShaft shaftB;
equation
  connect( shaftA.gear_con_surfs,
           shaftB.gear_con_surfs );
end GearBox;
```

Surfaces and surface sets are defined as Modelica *connectors* to allow for connect equations between surfaces of different bodies, i.e., to be able to model contacts between surfaces. Based on the object-oriented concept of information hiding, or encapsulation, Modelica puts restrictions on connect equations. That is, to keep a clean interface design Modelica does not allow for connect equations though several hierarchical levels, i.e., the following is prohibited in Modelica:

```
  connect( shaftA.gearWheel.con_surfs,
           shaftB.gearWheel.con_surfs );
```

To propagate the *GearSurfaceSet* connectors to the *GearBox* model level additional *GearSurfaceSet* connectors have been added to the *GearWheelShaft* model. These are internally connected to the *GearWheel* surfaces to make them accessible from outside. It is argued that from a multibody system contact point of view this is not a very good solution because:

1. In the most general case it requires to propagate all surfaces that can have contact though all higher levels in the model hierarchy.
2. It requires adjustment of all classes at higher hierarchical levels if the surface topology changes.
3. It makes class design difficult and error prone.

Note that the latest version of the Modelica multibody library addresses the problem of collision detection [21] between bodies by defining a global *contactData* table that that keeps information about bodies states. Collisions can be computed between any two bodies. Every body needs to be enabled for collision detection. It is then registered in the global contact-data table. This is a general approach for simple contact detection but not sufficient for detailed contact analysis because:

1. Detailed surface analysis often requires to split the surface of a body into several surface segments. This is needed to gain the necessary level of detail.
2. Not all contacts between two bodies have the same properties. Definition of contact properties for a certain contact between two specific surfaces or surface-segments is often desirable.
3. It is not efficient for detailed, computation intensive, contact calculations. Instead one wants to define the contacts of interest.

In many mechanical applications all possible contacts are known and can easily been defined. Contacts in BEAST are therefore part of the model hierarchy. Thus there exist several connection objects — one for each pair of bodies that might be in contact. BEAST uses a different approach because it puts much more emphasis on detailed contact analysis. Contact calculations in BEAST are computationally intensive and only contacts of interest are defined in the multibody system model.

Deep connection modeling, or the concept of connecting to objects at a lower hierarchical level, is in contrast to the Modelica *inner/outer* concept that allows to reference variables and objects at higher levels in the model instance hierarchy. One could argue that the deep connection concept conflicts with object-oriented encapsulation, but encapsulation does not prohibit access through multiple sub-levels in the object hierarchy as long as all interfaces at all levels are public accessible.

A combination of BEAST and Modelica models is of interest for the authors for different reasons. The main advantage is the possibility to have combined modeling capabilities, i.e., integrate Modelicas multi-physics capabilities into BEAST models or integrate BEAST components into Modelica models. Also, one could argue, that the problem of connecting different parts on different hierarchical levels is of more general nature, e.g., not only contacts in multibody systems are affected but other application areas as well. A more general solution for deep connections might thus be of interest for other Modelica users. Two possible approaches for extending Modelica with deep connection modeling capabilities are therefore presented here:

Global connectors that are propagated through all parent levels. This is a special connector type that is accessible through all higher levels of the model hierarchy.

Connector references that can be used to reference a connector at any hierarchical sub-level.

Global connectors can exist in conjunction with normal *connectors* to not break the concept of information hiding for normal connectors. They are accessible through multiple sub-levels by standard naming convention:

```
globalconnector GearSurfaceSet
  // Any code here
end GearSurfaceSet;

model GearWheel
  // Gear wheel contact surfaces
  GearSurfaceSet con_surfs;
end GearWheel;

model GearWheelShaft
  GearWheel gearWheel;
  Shaft shaft;
equation
end GearWheelShaft;

model GearBox
  GearWheelShaft shaftA;
  GearWheelShaft shaftB;
equation
  connect( shaftA.gearWheel.con_surfs,
           shaftB.gearWheel.con_surfs );
end GearBox;
```

Graphical modeling is an important aspect of Modelica. Graphical Modelica editors, e.g., MathModelica (Lite) or Dymola, should present global connectors visually on all higher model levels to simplify modeling. Visualization on higher levels should also be user selectable, i.e., enable/disable visual propagation of global connectors.

Connector references allow to reference any connector to make it accessible on different hierarchical levels in the model hierarchy. Reference variables are defined with the keyword *reference*. Reference variables are used just like any other variable. Here is the gearbox example:

```
connector GearSurfaceSet
  // Any code here
end GearSurfaceSet;

model GearWheel
  // Gear wheel contact surfaces
  GearSurfaceSet con_surfs;
end GearWheel;

model GearWheelShaft
  GearWheel gearWheel;
  Shaft shaft;
equation
end GearWheelShaft;
```

```
model GearBox
  GearWheelShaft shaftA;
  GearWheelShaft shaftB;

  reference GearSurfaceSet
    con_surfsA(shaftA.GearWheel.con_surfs);
  reference GearSurfaceSet
            con_surfsB(shaftB.GearWheel.con_surfs);
equation
  connect( con_surfsA,
           con_surfsB );
end GearBox;
```

This concept is harder to implement from a user interface point of view. One could have a reference browser that shows the complete model hierarchy where users can pick the connectors that they want to reference in the current model.

In conclusion one can argue that interface propagations as used in Modelica provides clean interface definitions with data encapsulation and information hiding. This approach works well for modeling of many systems but is not sufficient for convenient modeling of multibody contacts where global accessible interfaces are needed. However, above we briefly demonstrated that simple modifications to the Modelica language would allow modeling connections through multiple hierarchical levels.

5 Model libraries and model composition

Equation-based object-oriented multi-physics modeling and simulation environments, e.g., Modelica, allow class-based modeling using a number of object-oriented concepts, e.g., inheritance, encapsulation, and polymorphism. It is a language-based modeling approach where the class design often is part of the modeling process. Both pre-defined library classes and user-defined classes can inspected and modified, if needed. However, for many applications the user just assembles and connects instances of pre-defined library classes to create an application model, which is immediately instantiated.

On the other hand, specialized mechanical system simulation environments typically work with a set of predefined classes of objects. Here the underlying class design is hidden for the user, i.e., they are essentially black boxes. Changing such classes is possible only by modifying the underlying implementation in some programming language (e.g. C++, Fortran). For multibody system simulation environments such predefined classes are typically bodies, connections or joints between these bodies, and system boundary conditions, e.g., external forces. The modeling process is the assembly of these objects into a simulation model.

5.1 Model composition

Composition hierarchies are used to represent the structure of the system model. For example, a top-level gearbox object contains a housing and several gearwheel-shafts, where each of the shafts contains several gearwheels and bearings, etc.

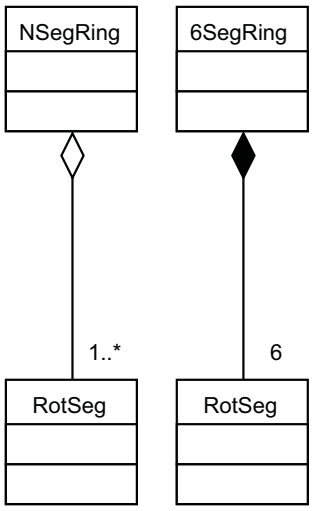
BEAST uses several factory design patterns [8] to assemble the simulation model. The factory pattern allows instantiation of objects at run-time. Factory based composition implies aggregation relations, see also Figure 5(a), where the factory is responsible for object instantiation and to create the model hierarchy. Note that there is no need to create new classes but the model is assembled from predefined classes.

In Modelica a class is a composition of other class instances, see also Figure 5(b). New classes are created, that possibly extending existing ones, whenever needed during the modeling process. Some people might see the connection equations in Modelica's as a weak form of aggregation. But the connection equation does not express containment.

The main difference between the two approaches is that aggregation together with assembly based on the factory design pattern allows for ad hoc model composition without the need to define new classes. The BEAST modeling language (BML), for instance, is used to store BEAST models. A simulation kernel reads the BML file and composes the model hierarchy on demand using different factories for bodies, segments, etc. This, however, requires that all possible classes of objects are already defined. Modelica composition hierarchies on the other hand are based upon class composition hierarchies, i.e., pure composition relations.

To summarize:

- In both cases, the user assembles a model from model components available in libraries, either hardwired libraries (in the BEAST case) or model libraries defined in the modeling language (in the Modelica case)



(a) *Aggregation relation.* (b) *Composition relation.*

Figure 5: *The difference between aggregation and composition in a bearing model. Aggregation relation allows for variable amount of bodies in a bearing model. Composition relation in Modelica implies a predefined amount of bodies.*

- In the BEAST case, the C++ implementation uses aggregation of object instances to create the assembly.
- In the Modelica case, the model editor generates a specialized class (using composition) which contains all components that the user specified during the assembly phase. This class is instantiated by the model compiler and run-time system before the simulation.

6 Conclusion

The application of object-oriented techniques to mechanical system modeling has been discussed in this paper. In 3D mechanical systems modeling and design, it is natural to be able to connect components whose surfaces are externally available. For example, the motor belonging to a car can be externally accessible in the 3D view, even though it can be regarded as an internal component of the car.

Two modeling approaches have been compared with special reference to Modelica and BEAST. Modelica has a broad application area, i.e., multi-physics system modeling including multibody system modeling as a special case. BEAST is used exclusively for mechanical system modeling, i.e., multibody system modeling, with special focus on detailed contact modeling. The comparison led to the following main observations:

- Modeling of contacts between mechanical parts requires accessible interfaces. Contacts in multibody systems are possible between any two bodies in the model that do not have any intervening object or boundary that prevents physical contact. For example, if the surface of a body is exposed to the outside world it can in principle be connected to any outside body, whereas a body which is hidden within a physical container (e.g. a box or a sphere) cannot be connected to outside bodies. If the simple solution of making all interfaces globally accessible is used, additional (geometrical) checking will be needed to prevent unphysical impossible contacts.
- Composition hierarchies can be created in several ways. One method of creating object hierarchies is object-oriented composition where the part objects are declared as belonging to a containing object. More flexible hierarchy composition can be achieved at runtime, e.g., adding or removing child objects on demand using flexible data structures such as linked lists to represent the containment relation.
- The modeling process is often very domain dependent. A general target domain of the simulation tool also requires a general modeling process. For example, general multi-physics modeling requires a high level of flexibility, i.e., class and interface design, while pure mechanical modeling can utilize predefined classes of objects.
- Graphical user interfaces are often domain dependent. For example, regarding 3D mechanical system modeling a user interface with support for three dimensional visual representations of the complete system is desirable.

The main difference, however, is data encapsulation in conjunction with contact modeling. Modelica requires strict interfaces with encapsulation of internal components, whereas BEAST allows connections to internal components which are visible in a 3D view, i.e., all bodies or their surfaces, which are exposed to the outside environment, must be accessible. Contacts in BEAST therefore have access to any child component at a lower level in the

model hierarchy. Model designers (BEAST users) create the contacts that are of interest for a particular system analysis and are responsible to avoid the creation of contacts between bodies whose surfaces are not physically accessible. Modelica on the other hand uses a more strict approach in deep connection modeling. Connections through multiple sub-levels in the model hierarchy require the connector to be propagated through all sub-levels.

Finally, different solutions have been briefly presented and discussed (in the form of possible Modelica extensions) to combine the advantages of both approaches, these are:

Global connectors that are propagated through all parent levels. This is a special connector type that is accessible through all higher levels of the model hierarchy.

Connector references that can be used to reference a connector on any hierarchical sub-level.

7 References

[1] Dahl, O.J. and Nygaard, K.: *How Object-Oriented Programming Started*, Dept. of Informatics, University of Oslo, http://heim.ifi.uio.no/ kristen/FORSKNINGSDOK_MAPPE/F_OO_start.html

[2] J. Rumbaugh and M. Blaha and W. Premerlani and F. Eddy and W. Lorensen: *Object Oriented Modeling and Design*, Prentice-Hall International Editions, 1991

[3] G. Booch: *Object-Oriented Analysis and Design with Applications*, Addison Wesley Professional, 1994

[4] I. Jacobson: *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley Professional, 1992

[5] M. Fowler: *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, Addison-Wesley Pub. Co., 2003

[6] Joel J. Luna: *Hierarchical Relations in Simulation Models*, Proceedings of the 1993 Winter Simulation Conference

[7] R. Sinha and V. Liang and C. Paredis and P. Khosla: *Modeling and Simulation Methods for Design of Engineering Systems*, Journal of Computing and Information Science in Engineering, March 2001, Volume 1, Issue 1, pp. 84-91

[8] E. Gamma and R. Helm and R. Johnson and J. Vlissides: *Design Patterns - Elements of Reusable Object-Oriented Software*, Addison Wesley, 1995

[9] P. Fritzson: *Object-Oriented Modeling and Simulation with Modelica 2.1*, Wiley-Interscience, 2004

[10] H. Elmqvist et al: *Dymola User's Manual - Version 5.0a* Dynasim AB, 2002

[11] Fritzson, P. and Nordling, P.: *Adaptive Scheduling Strategy Optimizer for Parallel Rolling Bearing Simulation*, HPCN Europe '99, Amsterdam, April 1999

[12] Stacke, L-E. and Fritzson, D. and Nordling, P.: *BEAST—a rolling bearing simulation tool*, Proc. Instn Mech. Engrs, part K, Journal of Multi-body Dynamics, 1999,

[13] Peter Fritzson, Peter Aronsson, Håkan Lundvall, Kaj Nyström, Adrian Pop, Levon Saldamli, and David Broman: *The OpenModelica Modeling, Simulation, and Software Development Environment.*, In Simulation News Europe, 44/45, December 2005. See also: http://www.openmodelica.org.

[14] P. Fritzson, and D. Fritzson: *Object-oriented Mathematical Modelling Applied to Machine Elements*, Computers & Structures, Vol. 51, No. 3, pp 241-253, 1994

[15] S. McMillan, and D.E. Orin, and R.B. McGhee: *A Computational Framework for Simulation of Underwater Robitic Vehicle Systems*, Journal of Autonomous Robots

[16] J.T. Cross, and I. Masters, and R.W. Lewis: *Why you should consider object-oriented programming techniques for finite element methods*, International Journal of Numerical Methods for Heat & Fluid Flow, Vol. 9, No. 3, 1999

[17] M. Otter, and H. Elmquist, and F.E. Cellier: *Modelling of Multibody Systems with the Object-Oriented Modeling Language Dymola*, "Nonlinear Dynamics", Vol. 9, pp. 91-112, 1996

[18] D. Fritzson, and P. Fritzson, and L. Viklund, and J. Herber: *Object-Oriented Mathematical Modelling – Applied to Rolling Bearings*, In Proc. of SCAFI-92, Amsterdam, Nov 5-6, 1992.

[19] D. Zimmer and E. Cellier: *The Modelica Multi-bond Graph Library*, Proceedings of the 5:th International Modelica Conference, Vienna, Austria, September 4-5, 2006

[20] M. Otter and H. Elmqvist and S.E. Mattsson: *The New Modelica MultiBody Library* Proceedings of the 3:rd International Modelica Conference, Linköping, Sweden, November 3-4, 2003

[21] M. Otter and H. Elmqvist and J. Díaz López: *Collision Handling for the Modelica Multibody Library*, Proceedings of the 4:th International Modelica Conference, Hamburg, Germany, March 7-8, 2005

[22] P. Fritzson, L. Viklund, J. Herber, and D. Fritzson: *Industrial application of object-oriented mathematical modeling and computer algebra in mechanical analysis.*, Technology of Object-Oriented Languages and Systems - TOOLS 7, pages 167-181. Prentice Hall, 1992

[23] Modelica Association: *Modelica - A Unified Object-Oriented Language for Physical Systems Modeling*, Language Specification, Version 3.0. 189 pp. Published at www.modelica.org. Sept 5, 2007.

[24] MathModelica: *http://www.mathcore.com/products/mathmodelica*, Mathcore AB

[25] Simantics – software platform for modelling and simulation: *https://www.simantics.org*