Proceedings of the 19th CIRP Design Conference – Competitive Design, Cranfield University, 30-31 March 2009, pp443

Invited Paper

Object-Oriented Simulation Model Generation in an Automated Control Software Development Framework

M.J. Foeken, M.J.L. van Tooren
Faculty of Aerospace Engineering, Delft University of Technology, Kluyverweg 1, 2629HS Delft, The Netherlands
{m.j.foeken, m.j.l.vantooren}@tudelft.nl

Abstract
The automated development of control software for mechatronic systems requires the integration of control models for design and verification purposes. To obtain high-fidelity models, unintended behaviour of the system must be taken into account which requires knowledge about the systems architecture and component interaction. Furthermore, integration of design and analysis tools into a meta-model framework is needed to exchange system information. This paper discusses the use of Modelica to organise libraries and model the behaviour of systems in this framework, and the need for a knowledge-based tool to automatically generate these models.

Keywords:
Mechatronics control software, Object-oriented modelling, Simulation, Integration framework, Knowledge-based engineering

1 INTRODUCTION

Nowadays, computers control industrial machines, information devices, aircraft and office equipment to name just a few. The development of such mechatronic products requires the collaboration of mechanical designers, electronic system engineers, aerodynamic engineers, and software engineers. Whereas software development for mechatronic systems in industry benefits from advances in tools and supporting systems, in general they still suffer from problems like a lack of integration across design domains, a lack of physical modelling, the need to handle irregular situations, and foremost, a lack of automation.

To attack these problems, a project named 'Automatic Generation of Control Software for Mechatronic Systems' was started to develop a set of prototype tools and an integration framework with which an interdisciplinary product development team can automatically generate control software for mechatronic systems. Figure 1 shows the framework with the set of eight tools that will be developed within the project, each represented as a white block.

The project envisions the use of a functional model as input to the control software generation process, specifying the required functionality of the system being developed. The 'Function Modelling' tool will create a formal representation of these functions, which will be used to generate the necessary behaviour based on qualitative reasoning methods. At the same time, the function model will enable the 'Mechatronic Feature Modelling' tool to generate the product definition by using mechatronic features, or function performers [1]. The behaviour description and the mechatronic feature model serve as an input for the mechanical embodiment and electrical system design. Combined with data from analysis tools as finite element method (FEM) or computational fluid dynamics (CFD) solvers which are often used in aerospace design, these form the basis for the control code and control model generation processes.

In Figure 1, these existing commercial software tools, like e.g. CATIA for mechanical CAD design or Fluent for CFD analysis, are represented by dashed-line blocks.

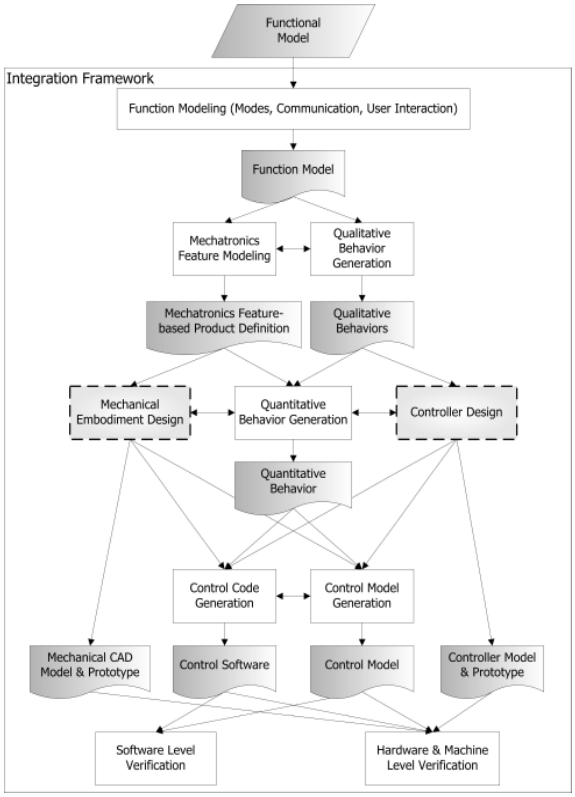

Figure 1: Systems architecture, with the white blocks representing tools being developed in the project. Dashed-line blocks correspond to existing, commercial software tools.

At the end of the design process, the generated code can be verified at software and hardware level, using either

the generated control models or the prototype hardware, respectively.

The integration of design and analysis tools in an automated framework will support a more concurrent software design process, in contrast to the sequential process often seen in practice, by automating the sharing of information across the design domains.

The framework in Figure 1 shows the control code generation and the control model generation processes in parallel. In this view, the 'control model' is defined as a model of the entire system minus the control software. This means that the control hardware, like for example a micro controller, might be part of the control model, if required.

The need for a control model generator in the software development framework arises from the wish to be able to verify the software at earlier stages of the development, before the real hardware has been built. A high-fidelity control model would enable the verification of the control software by using emulation and/or simulation methods, which on one hand partially eradicates the need for more expensive machine based verification, and enables software verification at earlier stages of the development on the other.

A second point of interest to the project is the integration of irregular situations and operating modes into the controller design development. Software development has to deal with irregular operation modes and abnormal situations, as well as regular modes like initialization, shutdown, maintenance and calibration. In terms of verification, this not only requires the controller, but also the control model to be adjustable to these situations.

To obtain a high fidelity model, not only the intended behaviour as needed to realise the required functionality, but also behaviour that was not anticipated on beforehand must be included. In that way, unexpected side effects of the design implementation can be discovered before prototype testing begins.

The methodology supporting the automated generation of control models and taking into account these requirements is the main focus of the current research. First, however, the implementation of modelling and simulation concepts during the controller design process will be discussed.

2 MODELLING AND SIMULATION

2.1 Object-Oriented Physical Modelling

As noted in [2] it is important to make a distinction between modelling and simulation. Whereas Websters Dictionary defines modelling as 'to produce a representation or simulation of,' the Oxford Dictionary defines it as 'to devise a mathematical model of.' More accurately, modelling can be defined as creating a simplification of reality based on physical principles, while simulation, in the broad sense, is an imitation of behaviour, for which mathematical models can be used.

The mathematical model normally used in controller design, being either feedback, sequential or hybrid, is often presented as a block diagram containing transfer functions, representing the particular behaviour of a system by means of state-space matrices, eigenfrequencies and damping coefficients, and mathematical operators. Often, linear models that are only valid at a nominal design point are used. Matlab/Simulink [3], the de-facto standard in controller design, fully supports this modelling paradigm.

However, taking into account the entire system design, there are more types of mathematical models available to verify whether requirements are met. FEM or CFD analysis are frequently applied methods to verify a design, each requiring a different kind of mathematical model based on other physical principles, and subsequently different kinds of simulation. An integrated parallel simulation might be attractive in terms of physical accuracy, the computational effort and time required for large scale CFD or FEM simulations makes the combination unsuitable for controller verification. Instead, combining these different types of simulations is normally done in a sequential order, with the results of the first used in the model of the second. With CFD methods, ranging from linearised potential flow to the Navier-Stokes equations, lift, drag and moment coefficients and stability derivates for a range of airspeeds can be derived, which are subsequently used in flight mechanics models during controller design. Typical properties that can be derived with these analysis tools are then used in controller design are collected in Table 1.

The block diagrams frequently used in controller design tools as Simulink are in principle nothing more than mathematical equations in a visual form, where the basic elements have no direct relation with the physical world. To model a real-life system with such elements, it is necessary for the designer to know:

- How to represent the expected behaviour of (a part of) the system in mathematical equations, and,
- In what form the equations must be written such that the input and output can be 'connected' to the equations of other parts.

An alternative to this signal based approach are bond graphs. Independent of the physical domain, the graphs consist of basic elements like junctions, resistors and capacitors to represent the flow of energy through the system. This 'physical modelling' already makes the modelling effort less prone to error. To move the viewpoint of the modeller from the equation level to the component level, bond graph elements can be combined into a model representing a physical component.

Parallel to the application physical modelling languages, the emergence of the object-oriented (OO) modelling paradigm and languages like Modelica [4], allowed for new model development methods. The OO modelling paradigm nicely suits the engineering view on the product definition, as models built from objects allow for a good mimic of the real world [5].

One has to keep in mind the difference between the use of objects as basic building blocks and the use of OO programming concepts like encapsulation, inheritance and polymorphism. Encapsulation is a method to 'hide' information, by concealing the internal methods of a class from objects that interact with it. The part that is visible to other objects is called the interface. Polymorphism is also related to interfaces, and is in general a method to ensure that different datatypes, e.g. integers or characters, can be handled by a consistent interface. Finally, (class) inheritance deals with the specialization of classes by introducing subclasses which inherit the attributes and methods of their parent class and subsequently add new attributes and methods of their own. The use of polymorphism and inheritance in modelling languages will be further discussed in Section 3.2.

Although a model can be built up from components that have object properties, the model language might not support these basic OO concepts. Reference [6] shows that bond graphs can be viewed as some kind of object-oriented physical modelling, and that OO languages like Modelica can be used to textually describe bond graphs. Furthermore, it is argued that the principle of encapsulation and inheritance makes the use of

component libraries and 'the building of large and complex engineering systems more safe.'

2.2 Related Research Object-Oriented Modelling

In the framework of the Open Library for Models of Mechatronics Components (OLMECO) project, [2] describes the architecture of an object-oriented library of reusable simulation models. The model structure that is suggested consists of three layers, being technical components, physical concepts, and mathematical relations. The choice for these three viewpoints based on the fact that each of them needs consideration when modelling. Figure 2 shows an adapted version of the top-level view of the library architecture, using UML notation. On the technical component level, the system decomposition is build of from various components which are part of a component class. These components can be represented by a conceptual physical description, which is build up from one or more bond graph elements, representing mathematical equations.

The same reference also discusses the need for a taxonomy of component classes to handle the complexity associated with large libraries. The kind-of relations do not restrict the structure to be tree-like, instead, a lattice structure can also be obtained.

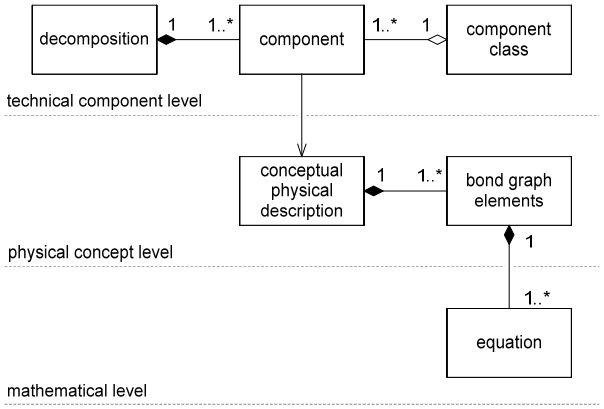

Figure 2: Top-level view of OLMECO library architecture, adapted from [2].

The use of objects is further extended in [7], which presents the concept of Composable Objects, combining form (CAD) and behaviour into a single object. By connecting these component objects to each other through their ports, it is possible to create both a system-level design description and a virtual prototype of the system. The interaction between components is port-based, and reconfigurable, so that components of different levels-of-detail are interchangeable. The relation between form and behaviour is given by a parametric description, ensuring that both remain consistent with each other. The behaviour of a mechanical design can be derived from constraints between parts [8].

Reference [9] discusses more about the use of an ontology for ports to be used for automatic model composition. With this ontology one can represent and verify compatibility between the ports in a connection, and reason to select the interaction models automatically. They also note that when connecting ports one must take into account the type of interaction taking place. Often, these interaction models depend on the parameters of both subsystems involved.

On the same topic, [10] presents a framework to capture the interaction in component based design. The system checks the compatibility of a component with a certain interaction type, preventing the coupling of incompatible components.

One of the issues when applying a component-based approach for this type of physical modelling is that the physical behaviour is not limited to the intended behaviour, which is often described in a single domain. An electric actuator, intended to transform electrical energy into mechanical energy, might produce an amount of heat that not only influences its own behaviour, but also that of other components. The ports of a component should therefore not be fixed nor restricted to only the intended connections, but must depend on the system's implementation.

3 CONTROL MODEL GENERATION

From literature, part of which is mentioned in the previous section, as well as from our own research, the following prerequisites have been recognised that enable automatic model generation in the context of the software development framework:

- It must be possible to build-up the system architecture from basic technical components.
- These components must have one or multiple representations in the physical modelling world. The taxonomy of the physical models should be based on component classes, amongst others.
- When connecting elements the port compatibility must be checked to prevent the coupling of incompatible elements.
- Not only the intended behaviour, but also 'secondary' behaviour must be recognised and included.
- For the required physical system parameters it must be possible to trace back to the providing design or analysis tools, or e.g. a database.

Of these, the first and last item on this list are related to integration into the development framework, whereas the other three concerned will have to take into account the methodology supporting the control model generation process. Previously, the authors have identified that using SysML [11] as a language for mechatronic system modelling supports the integration into the development framework by keeping an object centred view on the system [12]. Modelica, on the other hand, supports a component based modelling paradigm that can be used in combination with controller design methods and tools.

In the next sections, the above mentioned points will be further discussed. Apart from showing how SysML and Modelica support the automated generation of control models, the possible methodology to come from one to the other while taking into account these requirements will be introduced.

3.1 Mechatronic System Modelling

As stated in the introduction, the project considers functional modelling as part of the framework. Reference [13] considers the application of the Function-Behaviour-State modeller [14] as a basis for this functional model and discusses its use as a meta-modeller, facilitating the integration of other modelling tools. It shows that SysML is both powerful and flexible in representing the fundamental FBS concepts, and as such can be used to build meta-models.

The representation of the systems architecture in SysML is based on mechatronic or physical features. These features represent a physical component that performs a certain function, without specifying its mechanical embodiment beforehand and by that can be considered as a bridge between the function and the implementation

level. The level of abstraction of the functions and the mechatronic features is depending on the application domain as well as the amount of 'zoom', and as a consequence no common level of detail, nor a fixed amount of features, can be defined. However, in general the decomposition of the design continues until a physical relation between function, model and behaviour is known [15]. In Section 3.3 the discussion on these levels of detail will be extended to the use of high, middle and low level primitives as a possible solution for the automated control model generation process.

In terms of technical implementation, the use of SysML and the associated XML-based XMI language enable an easy mapping of the metamodel to other languages by means of one of the available XML processing techniques. The integration of SysML into the systems development process is also discussed in [16].

3.2 Polymorphic Physical System Modelling

The Modelica language has been designed to model large, complex and hybrid physical systems and is based on differential and algebraic equations. It supports non-causal and object-oriented modelling techniques, and as such stimulates the reuse of modelling knowledge. Although the language is text-based, the models can also be presented to the user as schematic block diagrams, each block representing a system component.

In general, a Modelica class contains a public declaration of parameters, variables and class instances, followed by the definition of equations and the connections between the instances. Connectors in Modelica can either be energy or information based, depending on the domain. Similar to bond graphs, the connection between physical components is achieved by specifying a flow and a non-flow variable for each connector, which when multiplied have the dimension of energy or power. The input to actuators and the output of sensors is a data stream.

The idealised dynamics model of a DC-motor can be represented by a combination of a voltage source, a resistor, an electrical ground, an electro-to-mechanical transformer, and an inertia element, see Figure 3. The parameters defining the behaviour of each of these separate elements, like the torque constant, are typically provided in the motor specification. Instead of showing the entire internal structure of the DC-motor's model, the 'DC-motor' component can be characterised by it's subelement parameters, as in Figure 4. The information embedded in the component is however not restricted to parameters only, but could also contain links to mechanical CAD drawing, including dimensions, masses, inertia, etc.

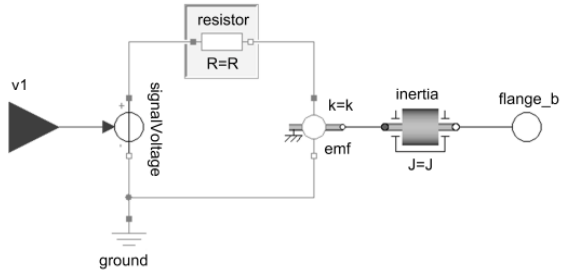

Figure 3: Idealised DC-motor model.

However, if a non-ideal model is required, taking into account e.g. friction and variable resistance due to heating, the DC-motor model has to be extended with additional elements, see Figure 5, requiring additional parameters as well. On the technical component level however, the element is still a 'DC-motor'.

The possibility to switch between models or elements of different complexity included in a bigger system model, named polymorphic modelling by De Vries [17][18], can be accommodated by ensuring that the ports of these model elements are identical. In object-oriented programming terms, coping with replacing objects is covered by the subtyping concept, which is based on the Liskov substitution principle [19].

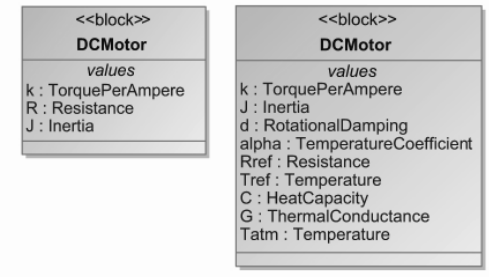

Figure 4: 'DC-motor' component in SysML, left for an idealised model, right with friction and variable resistor due to temperature effects.

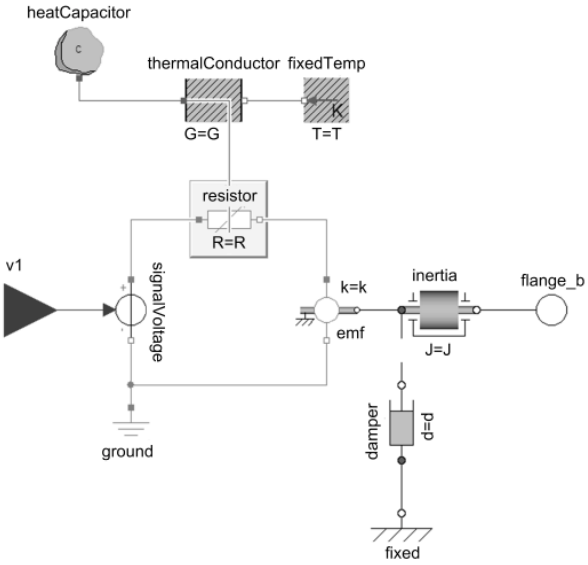

Figure 5: DC-motor model with rotational friction and variable resistor heating.

While class inheritance is a well known and often used concept in OO programming, the difference between class and interface inheritance, or nominal subtyping, lies in the fact that that the latter only describes when an object can be used in place of another, and does not describe the object's implementation [20]. With class inheritance, the methods are also inherited, which can subsequently be extended or possibly changed, depending on the language. For the subtyping concept inheritance is however not a requirement: an object can also be a subtype of another object without using interface inheritance, which is then called a structural subtype.

The type, or interface, is that part of the class that enables the substitution of one class with the other. In terms of physical modelling, this means that at least the ports or connectors of the component's physical description must be the same. However, characteristic parameters might also be part of the subtype, which makes that the DC-motor models in Figure 3 and 5 can be considered not to be subtypes.

The subtyping mechanism in Modelica is based on the object theory of Abadi and Cardelli [21]. The language specification defines a type or interface as 'the "essential" part of the public declaration sections of a class that is needed to decide whether A can be used instead of B' [4], where A and B are classes or components. At the same time, 'A is a subtype of B, or equivalently, the interface of A is compatible to the interface of B, if the "essential" part of the public declaration sections of B is also available in A'.

Due to the nature of the language, Modelica does not accommodate the re-declaration of the methods of a class (i.e. equations) when using the class inheritance mechanism, which prevents the full use of the subtyping concept when using class inheritance. Furthermore, when comparing Figures 3 and 5, the heating resistor component in Figure 5 is not a subclass of the normal resistor, as the resistance R is no longer a fixed parameter, but a variable. These two limitations prevent the creation of a natural specialization hierarchy based on only class inheritance relations.

This problem can be partially circumvented by using a so-called 'partial' model, such that one can create an methodless superclass from which multiple subclasses can inherit. In this way, grouping of components into component classes is still accommodated. A hierarchic class structure can be obtained by adding tagged information to each class, which can be done by using inheritance.

Altogether, Modelica enables both the use of OO modelling concepts to structure the libraries, as well as an object-based approach that supports the easy assembly and updating of the models itself.

The relation between elements on component and control model level is depicted in Figure 6.

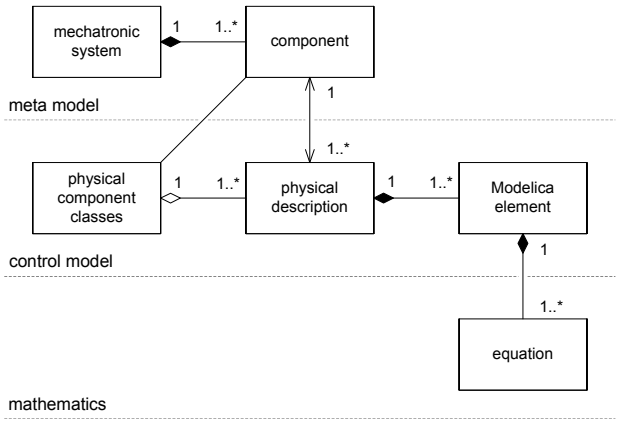

Figure 6: Relationships between components and elements at different viewpoints.

3.3 Model Generation

As becomes clear from the previous sections, a one to one mapping of components at different viewpoints is easily possible, with Modelica providing the capabilities to easily upgrade the physical model using object-oriented modelling techniques. However, if unintended behaviour due to component interaction in the model has to be included, a direct mapping method will not be sufficient, as additional modelling knowledge is required.

The application of knowledge engineering for the development of conceptual simulation models is discussed in [22]. In here, the focus is on how to capture, represent and organise the knowledge required for simulation modelling.

The use of expert knowledge in engineering applications to automate that part of the design and analysis process that is repetitive, non-creative and time-consuming can be supported by Knowledge Based Engineering (KBE) techniques and tools. In general, KBE tools implement rule based design, parametric CAD and object-oriented programming [23].

Though often the main focus is on creating new (CAD) designs, [24] gives an example of the use of domain specific modelling languages (DSL) as a base for KBE models that are not restricted to the geometrical domain. The method is applied to the design of wire-harnesses, which is both a geometrical as well as a conceptual problem, extending the application of KBE design methods beyond the geometrical domain. The basic building blocks, named high-level primitives (HLPs), represent classes containing sets of design rules that determine parameter values to instantiate objects. The collection of HLPs describing the system is called the product model and provides a parametric view on the system. Associated capability modules (CMs) describe processes that can be applied to the HLPs to generate certain views on the system, like e.g. a 3D or a finite element model.

The product models constructed using the current KBE systems like the ICAD system or Genworks' GDL [25] are object-oriented and based on general-purpose programming languages. GDL is a superset of ANSI Common Lisp, one of the two main dialects of Lisp, which is often used in artificial intelligence research.

The one on one mapping of these high-level primitives from the product model into the software model making up the DSL described in [23] and the use of CMs to obtain a specific view on the system provides a method to generate control models. The HLPs and CMs can be further split up in middle and lower level primitives, related to the various physical submodels that might be required. The decision on which physical submodels to use is based on the specifics of the system. The rules governing this decision can be formalised in such a way that information stored in or derived from the system meta-model can be used to generate the control model.

This so-called procedural knowledge describes the conditions under which processes or tasks are carried out, in contrast to conceptual knowledge, which deals with how objects are related to each other, among others. The knowledge on which a KBE tool is based is stored in a knowledge base, which can be split up in a process and a product composition part [26].

The development of an ontology underlying the knowledge base is a shared task for the entire research project: concepts defined in the meta-model, i.e. the features, must be related to the knowledge base on which the control model generation process is relying. As mechatronic systems come in quite different forms and sizes, the amount and type of features is fully dependent on the type of system.

Figure 7: 'Insight' quadrotor UAV [27].

4 APPLICATION

The concepts introduced in sections 3.1 and 3.2 will be exemplified by looking at the 'Insight' quadrotor UAV as

an example mechatronic application. Shown in Figure 7, the “Insight” is a quadrotor being developed at the Faculty of Aerospace Engineering of Delft University of Technology to perform indoors surveillance missions. The aircraft weighs 72 g, has a diameter of 30 cm and an endurance of 20 minutes while providing live streaming video [27].

The use of partial models defining only the type of the class can be illustrated by looking at the definition of the rotor components. For the calculation of rotor lift and drag multiple methods are available, the most simple only taking into account rotor speed in combination with a fixed lift and drag coefficient, while a more detailed model can take into account the local velocity field, air density and blade twist distribution. The blade’s inertia and the mechanical connectors are shared components defined in a partial model, while the algorithm to calculate the force and torque generated by the rotor are added in the full model. To capture aeroelasticity effects advanced algorithms for the calculation of the aerodynamic force distribution in combination with elastic structural elements are required, which is something that can not be easily modelled without specialist knowledge. The same holds for phenomena like rotor-rotor and rotor-ground interference, which requires the extension of the basic algorithm.

Figure 8 shows the three components and two connectors of the actuator assembly build up from standard Modelica library components, with the rotor and DC-motor components replaceable by other submodels.

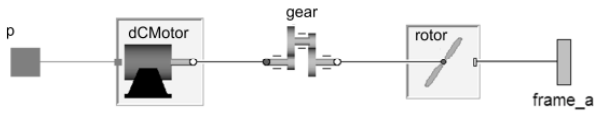

Figure 8: Actuator assembly for quadrotor UAV using standard Modelica library components.

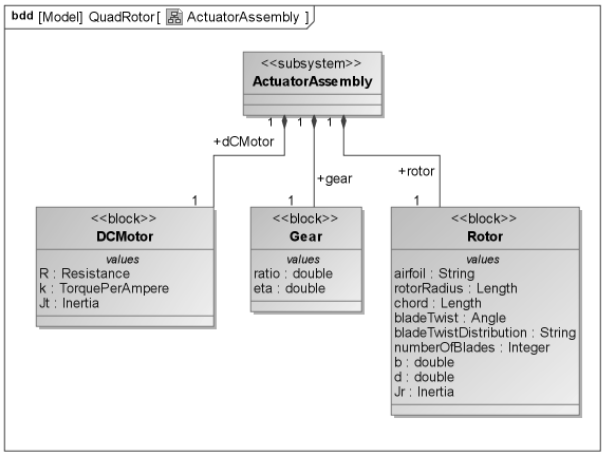

Figure 9: Actuator assembly hierarchical decomposition.

The one on one mapping of the DC-motor and gear component from the technical component view to a single physical description in the actuator assembly shows the problem when components are not only used for their main functionality, but, in this case, also as a load introducing part of the system. Instead of introducing force and torque back to the system via the gear and DC motor, by using standard library components they can only be introduced directly back into the system, or via separate 3D mechanics connectors through the gear and DC motor.

The systems meta-model in SysML consists of a hierarchical decomposition of the system in basic components. As in Figure 9, the actuator assembly consists of three components at the same level. For now, each of the components has only connectors in a single domain, such that they can only be connected in one way. These connectors are based on the type of energy they represent.

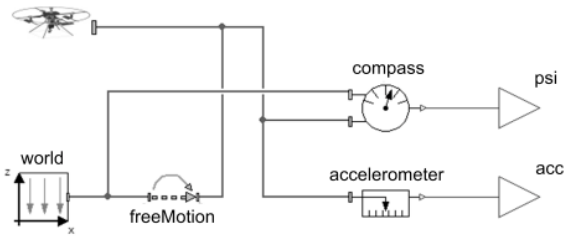

Figure 10: Sensors in the quadrotor assembly as used in controller development tools.

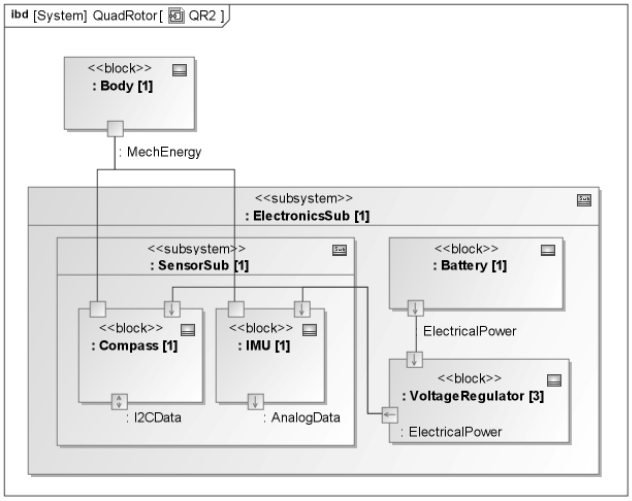

Figure 11: Part of sensor assembly, showing intended connections and outputs of technical components.

Additional problems related to both the mapping of components and the connections between components emerge when adding sensors and actuators. Where in normal controller design environments the output of ‘sensors’ are values of e.g. acceleration or angle, possibly with the addition of noise and delays (Figure 10), in reality sensors need an external power supply and the output is either an analog or digital signal which uses a specific type of software interface, as in Figure 11. The intended behaviour of the sensor can thus be viewed at different levels, each having a completely different physical representation. This difference has also to be taken into account in the control software generation process, which should take care for the extraction of the data from the analog or digital signals.

The amount of main technical components in the full model for the quadrotor is around 20, not taking into account the various parts of the structure. To map all of these into not only their intended, but also their not intended physical descriptions is already a task that needs expert knowledge in the various physical domains, as well as good insight in the system architecture itself.

Furthermore, even for a relatively small system like a UAV the amount of design and analysis data, like body mass, inertia, drag coefficients, lift and torque coefficients, sensor noise characteristics, etc. needed to

obtain a basic model is substantial. For industrial applications the amount of data easily becomes hard to manage, and tool and data integration by means of (meta-model) repositories is necessary.

5 SUMMARY

In relation to an automated control software development framework the need for a 'control model generator' has been discussed. An object-oriented physical modelling approach for the control model supports mapping of technical component or features at the meta-model level to physical descriptions in the control model view.

The Modelica language is well suited for polymorphic physical modelling as well as component library development; although the creation of class specialization hierarchies based on class inheritance alone is not possible.

To be able to include unintended behaviour in the control model, the interaction between components can not be fully predetermined. This dependence on the system implementation requires the use of expert knowledge in the form of formal rules. This is exemplified in the case study, which shows that a one on one mapping from a component based meta-model to a physical modelling description will not result in a high-fidelity model.

The application of knowledge based engineering techniques as a means to formalise and use this knowledge in an automated environment is considered. Using HLPs representing technical components as the basic building blocks of the system, the development of an ontology underlying the knowledge base is a shared task in the research project.

6 FUTURE WORK

The development of a mechatronic system meta-modelling concept based on features is the basis of the project's integration framework, on which various tools and associated methodologies rely. The ontology relating concepts in the various views on the system will enable the creation of a knowledge base which can be used to develop a control model generator application.

The use of a meta-model as the core of the development framework and the integration of design and analysis tools is the topic of separate research in the project, but is closely related due to the dependency of a possible tool on the information stored in the meta-model.

A knowledge-based tool able to generate control models based on information in the meta-model requires the acquisition of expert knowledge on physical modelling. Supported by methods able to determine interaction between components, the tool will be able to realise the requirements set in the introduction.

At a later stage, the addition of 3D models to the control model generation process will be added, to have a more 'physical' view on the system's behaviour during the verification process. This requires further integration of design tools, notably mechanical CAD, into the framework, such that not only design parameters, but the visual representation can be linked to the meta-model as well. Further application of KBE techniques to obtain and integrate this 3D representation seems a logical step, as parametric CAD is an integral part of most KBE platforms.

7 ACKNOWLEDGMENTS

The authors gratefully acknowledge the support of the Dutch Innovation Oriented Research Program 'Integrated Product Creation and Realization (IOP-IPCR)' of the Dutch Ministry of Economic Affairs.

8 REFERENCES

[1] Lutters-Weustink, IF, Lutters, F and Van Houten, FJAM, 2004, Mechatronic features in product modeling, the link between geometric and functional modeling?, Proceedings of International Conference on Competitive Manufacturing, Stellenbosch, South Africa: 125-130.

[2] Breunese, A, Top, JL, Broenink, JF and Akkermans, JM, 1998, Libraries of Reusable Models: Theory and Application, Simulation 71(1): 7–22.

[3] The MathWorks, 2008, MATLAB and Simulink, http://www.mathworks.com.

[4] Modelica Association, 2007, Modelica Language Specification - Version 3.0, http://www.modelica.org/documents/ModelicaSpec30.pdf.

[5] Sully, P, 1993, Modelling the World with Objects, Prentice-Hall, ISBN 0-13-587791-1, Englewood Cliff, New Jersey.

[6] Borutzky, W, 1999, Relations between Bond Graphs Based and Object-Oriented Physical Systems Modelling, International Conference on Bond Graph Modeling and Simulation, San Francisco, California, USA: 11-17.

[7] Paredis, CJJ, Diaz-Calderon, A, Sinha, R and Khosla, PK, 2001, Composable Models for Simulation-Based Design, Engineering with Computers 17: 112–128.

[8] Sinha, R, Paredis, CJJ and Khosla, PK, 2000, Integration of Mechanical CAD and Behavioural Modelling, IEEE/ACM International Workshop on Behavioral Modeling and Simulation, Orlando, Florida, USA: 31–36.

[9] Liang, V-C and Paredis, CJJ, 2003, A Port Ontology for Automated Model Composition, Proceedings of the 2003 Winter Simulation Conference, 1: 613–622.

[10] Lee, EA and Xiong, Y, 2001, System-Level Types for Component-Based Design, Lecture Notes in Computer Science 2211.

[11] Object Management Group, 2007, OMG Systems Modelling Language, http://www.omgsysml.org.

[12] Foeken, MJ, Voskuijl, M, Alvarez Cabrera, AA and Van Tooren, MJL, 2008, Model Generation for the Verification of Automatically Generated Mechatronic Control Software, IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications, Beijing, China: 275-280.

[13] Alvarez Cabrera, AA, Erden, MS and Tomiyama, T, 2009, On the Potential of Function-Behavior-State (FBS) Methodology for the Integration of Modelling Tools, CIRP Design Conference 2009, Cranfield, UK.

[14] Tomiyama, T and Umeda, Y, 1993, A CAD for functional design, Annals of CIRP'93, 42(1): 143-146.

[15] Schut, EJ, Van Tooren, MJL and Berends, JPTJ , 2008, Feasilization of a Structural Wing Design Problem, 49th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, Schaumburg, IL, USA.

[16] Friedenthal, S, Moore, A and Steiner, R, 2008, A Practical Guide to SysML: The Systems Modeling Language, Morgan Kaufmann, Burlington, Massachusetts, USA: 489-508.

[17] De Vries, TJA, 1994, Conceptual Design of Controlled Electro-Mechanical Systems, Ph.D. thesis, University of Twente, The Netherlands.

[18] De Vries, TJA, Breedveld, PC and Meindertsma, P, 1993, Polymorphic Modelling of Engineering Systems, International Conference on Bond Graph Modelling, San Diego, California, USA: 17–22.

[19] Liskov, B, 1987, Keynote Address - Data Abstraction and Hierarchy, OOPSLA '87: Addendum to the Proceedings on Object-Oriented Programming Systems, Languages and Applications, New York, NY, USA: 17–34.

[20] Gamma, E, Helms, R, Johnson, R and Vlissides, J, 1995, Design Patterns – Elements of Reusable Object-Oriented Software, 1st ed., Addison-Wesley.

[21] Abadi, M and Cardelli, L, 1996, A Theory of Objects, Springer, New York, NY, USA.

[22] Zhou, M, Son, YJ and Chen, Z, 2003, Knowledge Representation for Conceptual Simulation Modeling, Proceedings of the 2004 Winter Simulation Conference: 450–458.

[23] La Rocca, G and Van Tooren, MJL, 2007, Enabling Distributed Multi-Disciplinary Design of Complex Products: a Knowledge Based Engineering Approach, Journal of Design Research 3(5).

[24] Van der Elst, S and Van Tooren, MJL, 2008, Development of a Domain Specific Modeling Language to Support Generative Model-Driven Engineering of Aircraft Design, 26th Congress of International Council of the Aeronautical Sciences (ICAS) , Anchorage, Alaska, USA.

[25] Genworks International, 2008, General-Purpose, Declarative, Language, http://www.genworks.com.

[26] Stokes, M, 2001, Managing Engineering Knowledge: MOKA Methodology and Tools Oriented to Knowledge Based Engineering Applications. Professional Engineering Publishing Ltd.

[27] Insight Team, 2007, 'The Insight,' Faculty of Aerospace Engineering, Delft University of Technology, Delft, The Netherlands, D.S.E. final report.

Characteristic	Physical domain	Input and method
Lift / downforce	Aerodynamic	2D/3D geometry i.c.w. a fluid dynamics solver at certain conditions (speed, direction) to obtain lift coefficient.
Drag	Aerodynamic	2D/3D geometry i.c.w. a fluid dynamics solver at certain conditions (speed, direction) to obtain drag coefficient.
Stability derivates	Aerodynamic	2D/3D geometry i.c.w. a fluid dynamics solver at range of conditions (speed, direction) to obtain forces and moment. Further calculations to obtain derivatives.
Stiffness	Mechanical	3D geometry with material properties i.c.w. a finite element solver to obtain stiffness matrices.
Eigenfrequency	Mechanical	3D geometry with material properties i.c.w. a finite element solver to obtain eigenfrequencies.
Buckling strength	Mechanical	3D geometry with material properties, i.c.w.finite element solver to obtain strength
Heat capacity	Thermodynamic	Geometry with material properties i.c.w. finite element solver to obtain total capacity

Table 1: System characteristics and associated analysis tools.