Model-Driven Engineering: Combining Structure and Behaviour of Simulation Systems

Adrian Rutle[1] and Hans Georg Schaathun[1]

Høgskolen i Ålesund, postboks 1517, 6025 Ålesund
`{adru,hasc}@hials.no`

Abstract

In advanced marine operations safety and accuracy requirements must meet high standards. Training Simulators are the utilities crews use to train on expected and unexpected situations in a training environment that is similar to the real environment. Unfortunately, such simulators may be costly to build and maintain. Much of the cost is related to software components which have to be developed again when ships (or ship parts) are changed, or when new operations/scenarios are to be simulated. In this paper we outline a solution for the development of maritime training simulators based on model-driven engineering, which facilitates the specification of software at the model level, and automatically generate executable code that can be simulated and visualized in a training environment. We use the Eclipse modelling Framework to specify a model of a generic ship with links to components describing the behaviour of the physical parts comprising the ship. From this model we generate code for the desired programming languages, operating systems and computer architecture. We have chosen the language Modelica for describing the behaviour of the physical components.

1 Introdction

It is well-known that many features are moving from hardware to software, which is thus playing an increasing role in most engineering systems and other applications, at an increasing share of the cost. Many computer scientists have long recognised that software complexity is increasing much more rapidly than the improvements on programming languages and the methods to manage the complexity. Where software, measured in lines of code, is estimated to grow at a factor of ten per decade, very little has happened to programming languages since the introduction of object orientation around 1970.

In this project we are considering challenges and possible solutions within software for the maritime sector, in particular the development of training simulators for advanced marine operations. In the design of such training simulators, we see two major software challenges. Firstly, systems are composite and depend on the collaboration of programmers, electronic engineers, mechanical engineers, mathematicians and others. Each discipline will have to contribute to the code, but common-place languages of the day take a programming specialism to understand. Raising the level of abstraction, programming in a language which is closer to the shared mathematical syntax of the disciplines, we would reduce the risks of miscommunication, ease validation, and make the process less error-prone.

The second challenge is reuse of software components. The maritime industry is very fragmented, and every system will depend on modules from different vendors, each with their own favoured platform. Considering training simulators, we may need a propeller model, a hull model, and an ocean model from three different vendors. With different OS, language and API specification we have a problem. Again, modelling at a higher level of abstraction, a platform independent level, it should be easier to automate code generation for the platform of choice.

Our approach to handle these challenges is based on model-driven engineering (MDE), a development methodology where models are considered the first class artifacts in the development process. In MDE we shift the focus from code – the most important artefact in traditional development processes – to abstract models. These models are used in automated transformations to generate executable code. In this way, we achieve platform and architecture independence, contributing to reusable components that are easier to specify and to compose.

Our main focus is the development of maritime training simulators, but it is well worth noting that most of the theory and resulting framework will be equally applicable for virtual prototyping and simulation as part of product design. Virtual prototyping is a hot topic in the maritime industry. Where modelling and simulations tools are well known for individual components and subsystems, many questions must be answered before complete, composite ship systems can be virtually prototyped.

2 Training Simulator for Marine Operations

A *system* is an object or collection of objects where we want to study the properties [1]. It may be a building, a bridge, an aircraft, a ship, or a piece of software. A *model* of a system is (also) a system, representing the original system. Models may be abstracted or simplified to focus on particular perspectives that are relevant to specific experiments or analyses, thus leaving out details which may be less relevant. These experiments are carried out to find out how a system works in practice, before or after the system is built. An experiment performed on a model representing the original system is called *simulation*, and serve to demonstrate the dynamic behaviour of the system. A simulator model can *describe* an existing ship system, and at the same time *prescribe* a software system to simulate the behaviour of the ship system.

Often simulators communicate with a visualization module, which shows the model input, output, condition, etc in the form of a graph. In some simulations, it is necessary to visualize the system as close to the original, real system as possible; training simulators are such simulators since the crew must get a realistic experience from the training.

Now we outline a simplified ship simulator model. The idea here is to represent the software (which represents a real system/ship) in a high level model consisting of a structural system model connected to component models representing behaviour. The structure of the model consists of the following main components: Hull, Controller, Thruster Control, Actuator and Propulsor (see Figure 1). Of these components, Hull and Propulsor are models of physical systems, while Actuator is a model of the software used to compensate for the latency in real systems in order to make the simulation more realistic. Controller may be a model of software (e.g., Dynamic Positioning System) or of physical system (e.g., Joystick). The Thruster Control is a model of the software which communicates with Controller and Actuator.

Figure 1 shows the ship simulator model specified in the Eclipse Modeling Framework (EMF) [2]. The simulator software can be generated from this model; e.g., at any time, we can query the position of the ship, its speed and direction. In addition, we can perform queries on Actuator and Propulsor to retrieve information about these components' states.

The internal behaviour of the physical components of the model are specified in Modelica [1]. That is, the structural model is a skeleton which can be configured with different behaviour models. The components which communicate with Modelica models, for example, `PropulsorPhysicalModel` and `HullPhysicalModel`, delegate method calls from `Propulsor` and `Hull`, respectively, to the Modelica models representing these physical systems. Compatibility between the Modelica models and the rest of the system is facilitated by a common metamodel for physical units, including the *Système International* (SI), which is already sup-

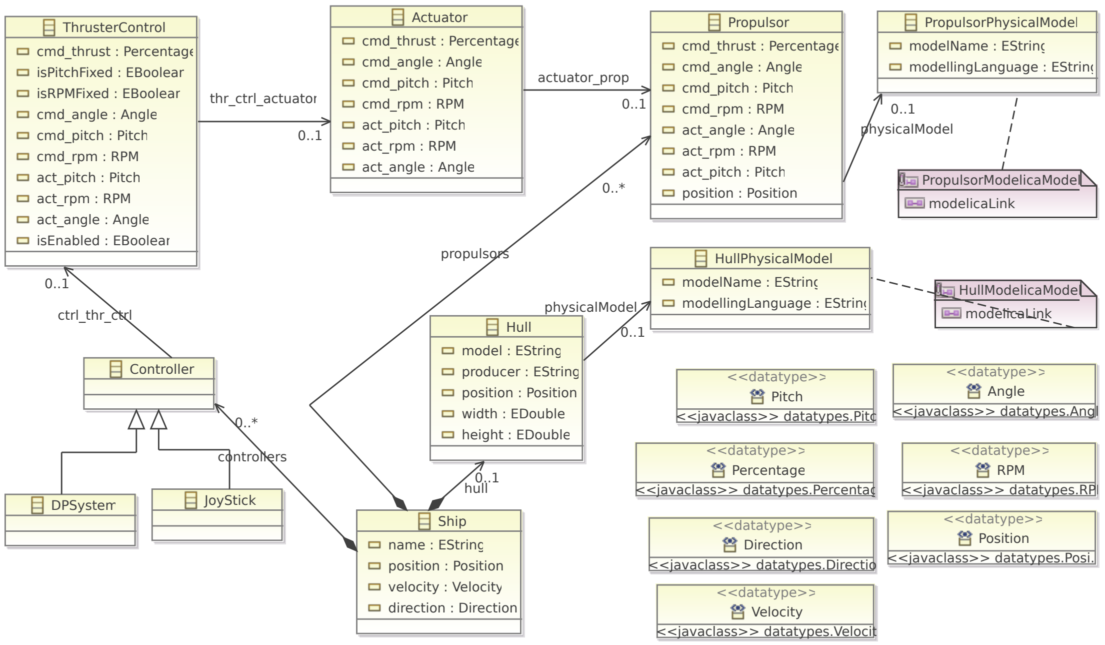

Figure 1: Ship model in EMF

ported by Modelica, as a subset. Some of these types are shown in the lower right hand part of the figure.

We focus here on one particular module of the ship metamodel, namely the one related to propulsion. This module consists of `Propulsor`, `PropulsorPhysicalModel` and `Propulsor-ModelicaModel`. The `Propulsor` has several static attributes, such as diameter, number of blades, its relative position to the hull, etc., in addition to various dynamic attributes such as commanded rpm (round per minute), actual rpm, commanded pitch, actual pitch, etc. The commanded version of an attribute is what the user sets as a goal value, while the actual version is the current value of the attribute in the system. When time passes the atual value will eventually be equal to the commanded value, however, there may be a latency until this convergence happen. The static attributes are unchanged for a specific propeller, however, the dynamic ones are changed under simulation.

`PropulsorModelicaModel` is a modelica model consisting of a set of parameters, variables, functions and equations, representing a propulsor's behaviour. The parameters of this model, such as diameter of the propulsor, number of blades, etc., are inherited from the structural model (see below). This model is responsible for calculating a force vector (speed and direction) based on the static and dynamic attributes of the propulsor. The force vectors from all propulsors, combined with other forces like waves and wind, are used by the simulator to calculate the overall behaviour of the ship. Defining the modelica model is a task for domain specialists with expertise in hydrodynamics, mechanical engineering, etc.

The `PropulsorPhysicalModel` is used to declare mappings between parameter and variable names of `Propulsor` and `PropulsorModelicaModel`. Another functionality of the `Propulsor-PhysicalModel` is to achieve a sort of low coupling between the structure of the system and the model which represents the behaviour, so that one can plug in different behaviour models without the need to change the structural model.

Next step is to generate runnable code from these models. The compiler system is still work in progress, and it is a complex thing, because it will have to generate code from both the system model from EMF and the component models written in modelica, and make sure that all the generated components are compatible.

Our preferred target language is Java, but this is not material for the work. We will design new software components for the compiler system with loose coupling so that other target languages can be introduced with relative ease at a later stage. The choice of Java is motivated by the Java virtual machine (JVM) which provides platform independence, and it is more established than other languages with compilers for JVM.

At the system or structural level, we can use existing technologies such as Java Emitter Templates (JET) or Xpand to generate Java code. When it comes to the modelica models, there are different options. DeltaTheta provides a product called Modelica SDK to compile modelica models into libraries for other programming languages, but a single, highly priced, closed source, proprietary solution is not satisfactory. The risk of discontinued support is high, and one will probably have to live with any flaws or missing features. Independently of this project, an open source competitor to DeltaTheta is called for.

The most viable option, from our point of view, is to modify and extend one of the existing open source compilers for modelica. Unfortunately, the tools that we are aware of generate code to be run inside their own simulation environment. In order to produce flexible code which can be integrated with (say) interactive software, rather extensive modifications are needed.

We have chosen the OpenModelica Compiler (OMC) as our starting point. Recently, parts of the compiler have been decoupled, with the introduction of the SUSAN template language, which simplifies the addition of support for new target languages. Although it is rarely used, the type information from the quantity and unit attributes is preserved, and be used to produce an API which is strictly typed using types such as SI units drawn from the application domain. There is even a prototype java code generator (JCG), but it is (still) incomplete. Thus we have both to complete the JCG to support the full modelica language, and adapt it to produce a more generic API which can be used in various kinds of software, including non-simulation tools.

One important aspect is that the generated java code must be compatible with that generated from the structural model, and it is useful if the structural model can dictate the API of the generated code. Thus, the SUSAN templates may have to be configured based on the structural model (one way or another) so that parameter and variable names remain the same as defined in the structural model. Moreover, the values of the static and dynamic attributes from the structural models will be used as parameters and variables in the modelica model, respectively. That is, static attributes, e.g. `diameter` in `Propulsor`, will be mapped to parameters, e.g. `diameter` of `PropulsorModelicaModel`, in the `PropulsorPhysicalModel`, typically by using `map(diameter, diameter);`. Dynamic attributes, e.g. `act_rpm` in `Propulsor`, are mapped to variables, e.g. `act_rpm`, of the corresponding modelica model. This is done for all attributes, and could be automatically generated if each attributes of `Propulsor` have a corresponding parameter or variable in the `PropulsorModelicaModel`. If the providers of the modelica model did not define the same names, these mapping could be used accordingly to reflect the right mappings. The information needed to do this configuration is available in the `PropulsorPhysicalModel`. We are not yet able to tell how challenging the necessary adaptions to OMC are going to be.

A natural question to ask, is why we only use modelica for individual components, and do not compose the complete system in modelica. There are two reasons for this. Firstly, modelica would normally accumulate all the constituent models into a single system of equations. If the simulator is sufficiently complex, the resulting equations may give a computationally intractible problem. Our approach gives us a way to split the system into units of compilation in a structural, high-level model. Secondly, the different constituent models are unlikely to come from the same vendor, and we may not be able to entise every vendor to use modelica. The high-level modelling framework can straight-forwardly be extended to support for instance FMI or *ad hoc* API-s.

3 Related Work

Graphical Modelica Modelling Language (ModelicaML) is a UML profile for Modelica which enables an integrated modelling and simulation of system requirements and design [3]. ModelicaML combines UML and SysML's standardized graphical notations for system modelling with Modelica's strength in modelling and simulation of physical systems. It makes possible to create executable specification and analysis models that can be simulated in discrete and continuous time. SysML4Modelica is another UML profile that is designed to standardize the ralationships between OMG's technologies and Modelica [4]. This profile can be used to annotate SysML elements with Modelica specific information. SysML4Modelica comes with a transformation – SysML-Modelica – that generates Modelica code from SysML design models. These profiles and the transformation SysML-Modelica is all based on the OMG's well-known technologies.

The Modelica community has specified the Functional Mock-up Interface (FMI) [5] to enable model exchange and deployment across tools. This API is C-oriented (as Modelica otherwise,) thus some machine architecture, operating system and language dependence is expected.

4 Conclusion and Future Work

Currently we are investigating the usability of the DSML for specification of single systems such as a ship or a crane. In future, the DSML will be extended to enable specification of training scenarios involving a various number of ships, cranes, winches, etc. Thus, we will generalise the approach from building models by connecting different components to building scenarios by connecting various models.

In this article we have outlined a solution for the development of maritime training simulators based on the MDE. The approach facilitates the development and composition of software components on the model level, and later to generate executable code that can be simulated in a training environment. We have specified a model for ship with links to Modelica models describing the behaviour of the physical components.

References

[1] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Press, 2004.

[2] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework 2.0 (2^{nd} Edition)*. Addison-Wesley Professional, 2008.

[3] Wladimir Schamai, Peter Fritzson, Chris Paredis, and Adrian Pop. Towards unified system modeling and simulation with modelicaml: Modeling of executable behavior using graphical notations. In *Proc. of the 7th Int'l Modelica Conference*, pages 612–621. Link´oping University Electronic Press, 2009.

[4] Christiaan Paredis et al. An overview of the sysml-modelica transformation specification. In *2010 INCOSE International Symposium*, July 2010.

[5] Torsten Blochwitz et al. The functional mockup interface for tool independent exchange of simulation models. In Christoph Clauß, editor, *Proc. of the 8th Int'l Modelica Conference*, pages 105–114. Link´oping University Electronic Press, March 2011.