CRANFIELD UNIVERSITY

**LEONARDO POETI**

# AN OBJECT-ORIENTED MODELLING METHOD FOR EVOLVING THE HYBRID VEHICLE DESIGN SPACE IN A SYSTEMS ENGINEERING ENVIRONMENT

SCHOOL OF ENGINEERING

PhD THESIS

CRANFIELD UNIVERSITY

SCHOOL OF ENGINEERING

PhD THESIS

Academic Year 2010-2011

LEONARDO POETI

# An Object-Oriented Modelling Method for Evolving the Hybrid Vehicle Design Space in a Systems Engineering Environment

Supervisor: Dr. James Marco

January 2011

# Abstract

A combination of environmental awareness, consumer demands and pressure from legislators has led automotive manufacturers to seek for more environmentally friendly alternatives while still meeting the quality, performance and price demands of their customers. This has led to many complex powertrain designs being developed in order to produce vehicles with reduced carbon emissions. In particular, within the last decade most of the major automotive manufactures have either developed or announced plans to develop one or more hybrid vehicle models. This means that to be competitive and offer the best HEV solutions to customers, manufacturers have to assess a multitude of complex design choices in the most efficient way possible. Even though the automotive industry is adept at dealing with the many complexities of modern vehicle development; the magnitude of design choices, the cross coupling of multiple domains, the evolving technologies and the relative lack of experience with respect to conventional vehicle development compounds the complexities within the HEV design space.

In order to meet the needs of efficient and flexible HEV powertrain modelling within this design space, a parallel is drawn with the development of complex software systems. This parallel is both from a programmatic viewpoint where object-oriented techniques can be used for physical model development with new equation oriented modelling environments, and from a systems methodology perspective where the development approach encourages incremental development in order to minimize risk. This Thesis proposes a modelling method that makes use of these new tools to apply OOM principles to the design and development of HEV powertrain models. Furthermore, it is argued that together with an appropriate systems engineering approach within which the model development activities will occur, the proposed method can provide a more flexible and manageable manner of exploring the HEV design space.

The flexibility of the modelling method is shown by means of two separate case studies, where a hierarchical library of extendable and replaceable models is developed in order to model the different powertrains. Ultimately the proposed method leads to an intuitive manner of developing a complex system model through abstraction and incremental development of the abstracted subsystems. Having said this, the correct management of such an effort within the automotive industry is key for ensuring the reusability of models through enforced procedures for structuring, maintaining, controlling, documenting and protecting the model development. Further, in order to integrate the new methodology into the existing systems and practices it is imperative to develop an efficient means of sharing information between all stakeholders involved. In this respect it is proposed that together with an overall systems modelling activity for tracking stakeholder involvement and providing a central point for sharing data, CAE methods can be employed in order to automate the integration of data.

*Dedicated to my parents Americo and Alida,*
*for being my source of encouragement and inspiration.*

# Acknowledgements

Firstly, I would like to thank Dr. James Marco, Prof. Nicholas Vaughan and all the staff of the Department of Automotive Engineering for their guidance, advice and patience. I would also like to express my gratitude to Will Stuart from Jaguar Cars Ltd. for the helpful discussions and insights he was able to provide me with regards to the automotive industry. I am most grateful to the Morgan Motor Company for the learning opportunity provided to me by way of the LifeCar2 project, and to the staff at Claytex for their assistance and response to the many questions I asked while developing models with Dymola.

I would also like to thank my friends and colleagues at Cranfield for the encouragement and support they offered, without which maintaining focus and sanity over these past years would not have been possible. In particular, those with whom I have shared both office and home: King Tin Leung, Ciprian Antaloae and Guido Monterzino, for the many enlightening discussions and shared emotions.

Finally, I would like to thank my family for always believing in me and making sure I never lost sight of my objectives during the tough moments.

# Contents

# List of Figures

# List of Tables

# Code Listings

# Nomenclature

| | |
|---|---|
| 2WD | Two Wheel Drive |
| 4WD | Four Wheel Drive |
| AFPM | Axial Flux Permanent Magnet |
| CAD | Computer Aided Design |
| CAE | Computer Aided Engineering |
| CFD | Computational Fluid Dynamics |
| CoG | Centre of Gravity |
| COTS | Commercial Off-The-Shelf |
| CSSL | Continuous System Simulation Language |
| DAE | Differential Algebraic Equation |
| DASSL | Differential Algebraic System Solver |
| EV | Electric Vehicle |
| FCV | Fuel Cell Vehicle |
| FMI | Functional Mock-up Interface |
| FMU | Functional Mock-up Unit |
| FWD | Front Wheel Drive |
| gPROMS | general PROcess Modeling System |
| HEV | Hybrid Electric Vehicle |

| | |
|---|---|
| HIL | Hardware-In-the-Loop |
| HVAC | Heating, Ventilation and Air Conditioning |
| ICE | Internal Combustion Engine |
| IGBT | Insulated-Gate Bipolar Transistor |
| IP | Intellectual Property |
| $LiFePO_4$ | Lithium Iron Phosphate |
| NEDC | New European Driving Cycle |
| NVH | Noise, Vibration and Harshness |
| ODE | Ordinary Differential Equation |
| OMG | Object Management Group |
| OOM | Object-Oriented Modelling |
| PPS | Peaking Power Source |
| RWD | Rear Wheel Drive |
| SED | Smart Electric Drives |
| SOC | State Of Charge |
| SUV | Sport Utility Vehicle |
| SysML | Systems Modeling Language |
| UDDS | Urban Dynamometer Driving Schedule |
| UML | Unified Modeling Language |
| VHSIC | Very High Speed Integrated Circuit |
| VHDL | VHSIC Hardware Description Language |
| VHDL – AMS | VHDL for Analogue and Mixed-Signals |
| VMA | Vehicle Model Architecture |

# List of Symbols

| | |
|---|---|
| $\eta_{mesh}$ | mesh efficiency |
| $\lambda_A$ | stoichiometry scaling factor for air |
| $\lambda_H$ | stoichiometry scaling factor for hydrogen |
| $\hat{\mu}$ | peak friction coefficient |
| $\mu_{roll}$ | rolling resistance coefficient |
| $\mu$ | coefficient of friction |
| $\omega_{rel}$ | shaft relative angular velocity |
| $\omega_m$ | nominal motor speed |
| $\omega_n$ | resonant frequency |
| $\omega$ | angular wheel velocity |
| $\omega r_w$ | wheel rolling velocity |
| $\phi_{in/out}$ | input/output rotation angle |
| $\phi_{rel}$ | relative rotation angle |
| $\rho_a$ | air density |
| $\sigma$ | tyre slip ratio |
| $\tau_{TR}$ | tractive torque |
| $\tau_{dmd}$ | demanded braking torque |
| $\tau_{friction}$ | friction brakes torque demand |

| | |
|---|---|
| $\tau_{friction}$ | friction torque |
| $\tau_{in/out}$ | input/output torque |
| $\tau_m$ | machine torque |
| $\tau_{max}$ | maximum acceptable regenerative torque |
| $\tau_{regen}$ | actual regenerative torque demand |
| $\tau_{req}$ | regenerative torque request |
| $a$ | acceleration |
| $A_f$ | vehicle frontal area |
| $c_{f,r}$ | front,rear axle suspension damping |
| $C_w$ | aerodynamic drag coefficient |
| $\mathrm{C_{cell}}$ | cell capacitance |
| C | ultracapacitor string capacitance |
| $d$ | shaft damping constant |
| $\mathrm{E_{uc}}$ | ultracapacitor energy storage |
| $F$ | force |
| $\mathrm{F_{AD}}$ | aerodynamic drag force |
| $\mathrm{F_{RR}}$ | rolling resistance |
| $\mathrm{F_{sw}}$ | switching frequency |
| $\mathrm{F_{TR}}$ | tractive force |
| $\mathrm{F_X}$ | driving force |
| $\mathrm{F_Z}$ | load force |
| $i_{gear}$ | gear ratio |
| $\mathrm{I_a}$ | nominal armature current |

| | |
|---|---|
| $I_{charge}$ | maximum battery charging current |
| $I_{supply}$ | supply current |
| $I_{sw}$ | switching current |
| $\hat{I}_{uc}$ | maximum ultracapacitor current |
| $J_r$ | electrical machine rotor inertia |
| $J_w$ | wheel inertia |
| $k$ | shaft torsional stiffness |
| $k_\phi$ | torque constant |
| $k_{f,r}$ | front,rear axle suspension stiffness |
| $k_{regen}$ | regenerative braking weighting factor |
| $L_a$ | armature inductance |
| $m$ | mass |
| $m_{f,r}$ | front,rear axle vehicle mass distribution |
| $m_{H_2}$ | hydrogen mass |
| $m_v$ | vehicle mass |
| $MF_{air}$ | air mass flow rate |
| $MF_{H_2}$ | hydrogen mass flow rate |
| $n_{cell}$ | number of ultracapacitor cells |
| $P_{comp}$ | fuel cell compressor power |
| $P_{cond}$ | conduction power losses |
| $P_{convLoss}$ | converter power losses |
| $P_{fc_{net}}$ | net fuel cell power |
| $P_{fc_{total}}$ | total fuel cell power usage |

| | |
|---|---|
| $P_{sw}$ | switching power losses |
| $r_w$ | wheel radius |
| $R_a$ | warm armature resistance |
| $R_b$ | battery internal resistance |
| $u(t)$ | input variables |
| $v$ | vehicle velocity |
| $V_a$ | nominal armature voltage |
| $V_b$ | battery internal voltage |
| $V_{bat}$ | battery pack voltage |
| $V_{boost}$ | ultracapacitor setpoint voltage |
| $V_{cell_{max}}$ | maximum ultracapacitor cell voltage |
| $V_{IGBT}$ | constant voltage drop across the IGBT |
| $V_{max}$ | maximum ultracapacitor/bus voltage |
| $V_{min}$ | minimum ultracapacitor bank voltage |
| $V_t$ | battery terminal voltage |
| $V_{uc}$ | ultracapacitor bank voltage |
| $x(t)$ | state variables |
| $\dot{x}(t)$ | time derivative of x(t) |
| $y(t)$ | output variables |

# Chapter 1

# Introduction

## 1.1 Background

Increasing fuel prices, pressure from legislators to move away from fossil fuel dependencies and the international drive to prevent global warming, places pressure on vehicle manufacturers to develop "greener" vehicles. In this context, "greener" refers to vehicles with reduced fuel consumption and carbon emissions. One way in which automotive manufacturers are complying with these new demands is through the development of competitive Hybrid Electric Vehicles (HEVs). In order to gain market acceptance and efficiently commercialise HEVs, vehicle manufacturers need to continuously diversify and change in order to secure competitive advantage [1]. Further, to offer variety to the market, manufacturers need to asses the merit of many different hybrid powertrain topologies in a fast and cost effective manner while maintaining quality and innovation [1].

In addition to the variety of HEV topologies, HEV powertrains make use of more components than conventional vehicles [2]; particularly electric components such as energy storage devices, controllers, power electronics and electric machines. These additional components dynamically interact with the conventional mechanical components. Therefore there are a much larger number of parameters and combinations which need to be analysed in order to optimise a particular powertrain design. Analysis of such a system is difficult due to its multidisciplinary nature and testing through prototyping can be both costly and time consuming [3]. The availability and use of computer modelling and simulation tools is a key component in studying

the complexities of HEV powertrain design [4].

Traditionally, many computer simulation models are built during the design life-cycle of a new vehicle. With different models being built for different purposes such as models to study fuel economy, models for controller evaluation, models for optimization or models for design exploration [3]. This leads to the creation of many models of the same system which all need to be maintained and managed. As shown in Section 2.2.2, when traditional block oriented models are employed, various versions of each model may be required in order to allow for different levels of fidelity and different causalities.

For conventional vehicle design, even though the complexity of modern vehicles is increasing, many of the systems being modelled are based on well known and understood previous designs and architectures [5, 6]. The availability of existing validated models and expert knowledge of the systems involved is an essential part of the initial development of new designs. With HEVs, however, there is a much smaller and limited amount of prior knowledge and many designs are still at a conceptual phase, meaning that many changes may still occur to the system.

According to Schyr and Gschweitl [7], the primary reasons for looking at new ways of developing HEV powertrains are:

- Reducing development time,
- Exploring new designs,
- Increased complexity,
- Higher expectations for reliability, and
- Reducing the amount of prototype testing needed.

In the software development domain, it has been shown that the time and effort required to build a system increases exponentially with the size and complexity of the design [8]. It can be argued that the same applies to modelling. The use of traditional modelling methods for exploring different HEV design options, risks reaching its limitations as far as managing the complexity of the design space is concerned [9]. Further, it will be necessary to provide sufficient flexibility to the designer to explore many possibilities in as short a time as possible. This leads to

a model maintenance and management problem when considering the number of models that such a process will generate and the potential number of different users that would be involved.

## 1.2 Problem Statement

HEV powertrains are complex systems with many configurations that combine electronic, electrical, chemical, mechanical and control components. Vehicle system analysis during the design process requires a multi-domain modelling approach due to the increasing interaction between mechanical and electrical components in modern vehicles [10]. Modelling and simulation tools capable of handling the complex interactions between these components are needed to explore designs before building physical prototypes [7, 11].

It is argued that a design environment for HEV powertrains should exhibit the following four characteristics [11]:

1. Modularity – allows for various systems to be built by combining different components. Ideally a library of components should exist that can be used and extended to build any powertrain configuration. The property of modularity becomes a requirement of any system once the complexity increases above an easily manageable point. With the objectives of producing a variety of HEVs while reducing the development time and cost, the ability to reuse previous design work is required.
2. Flexibility – to allow for varying levels of detail for each component and the use of components for both forward and reverse dynamics studies. Ideally the user should be able to select between simple steady-state components and detailed dynamic components or optionally create a component with the level of detail required. These components should be easily interchangeable. This is particularly important in the case of HEVs since a variety of technologies and topologies are still being investigated and the best practice still has to be established.
3. Multi-domain – since HEV powertrain design incorporates multiple systems such as mechanical, chemical, electrical and control systems. The simulation environment needs to be able to represent each of theses systems with the

same level of fidelity and accuracy.

4. Mathematical Rigour – since ultimately the true value comes from the correctness of simulation results which depend on the accuracy of the mathematical techniques employed. The methods used should be robust and efficient for all the physical domains being modelled in order to achieve accurate and fast results. Once again this is necessary for all design environments but since HEV design incorporates more multi-domain interactions, the possibility of "stiff" systems is much higher. Further, mechatronic systems such as HEVs exhibit both discrete events and continuous time processes, thus the numerical methods used to solve these systems should be particularly advanced [12].

Within the automotive industry, the development of HEV powertrain technologies is a relatively new practice; there is limited practical experience and building prototypes is expensive. Modelling and simulation is an ideal way to study and gain experience in this area [11, 13]. Frontloading the product development process allows for design validation and optimization during the early development stages through the use of software prototypes of HEVs and their components. This allows for important design decisions to be made at an early stage in the development process, rather than at later stages where changes could be more costly and problematic [7]. Obviously the benefit of such an approach is directly proportional to the amount and accuracy of data that is available during the initial modelling stages. Often valuable empirical data only becomes available during later testing stages. Therefore, it is necessary for the model development process to have an iterative nature that provides opportunity to modify an adapt models as data becomes available and design decisions are made.

The author acknowledges that the problem of model complexity is by no means new to the modern automotive industry. However, it is argued that this problem is compounded in the much larger and less explored HEV design space.

## 1.3 Aims, Objective and Scope

The main objective of the research undertaken in this Thesis is to propose a modelling method that can be used to more easily explore the multiple and complex powertrain design options that are possible within the area of HEV development.

Further, this development method should:

A. take advantage of OOM principles for more efficient development, and

B. be well suited for use within a systems engineering framework in order to provide better stakeholder communication, data sharing, maintenance and overall management.

In order to validate this objective the following aims are set out:

1. Apply object-oriented principles to the development of a powertrain modelling library.
2. Validate the object-oriented models created via comparison with previously validated Simulink models.
3. Use the proposed modelling method to investigate a new real world problem.

It is important to note that the main focus of the work presented in this Thesis is on the benefits of employing an object-oriented development method within the specific area of HEV powertrain design, and how this can provide further benefits when integrated within the systems engineering life cycle processes. It is outside of the scope of this work to define the specific systems engineering framework, methodology or tools that should be employed by an automotive manufacturer.

## 1.4 Thesis Structure

This Thesis is structured in order to help the reader understand the intersection of the research areas presented in Figure 1.1.

Chapter 2 discusses why the HEV design space is more complex than that of conventional vehicles and highlights features that a modelling environment should provide in order to effectively explore many feasible HEV designs.
Chapter 3 presents relevant features of object-orientation that promote the simplified development of complex systems.
Chapter 4 proposes a modelling method for the structured development of HEV models that exploits the cited advantages of object-oriented modelling languages.
Chapters 5 and 6 present two example case studies that illustrate the use of this

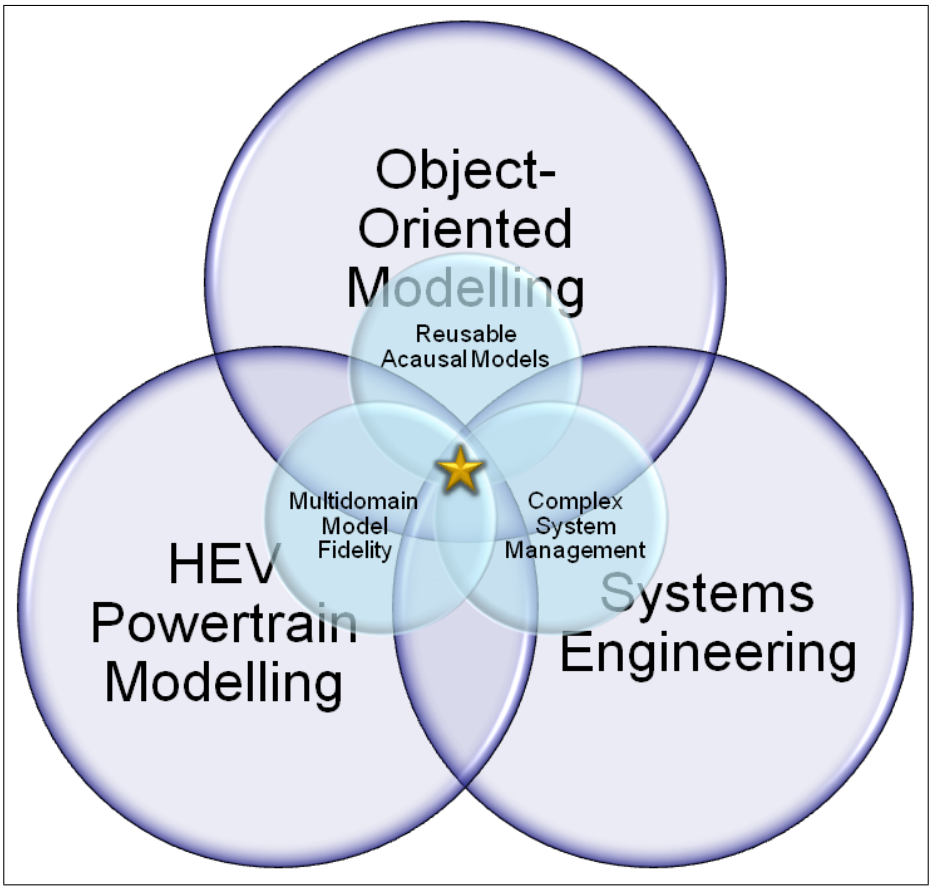



Figure 1.1: Diagram depicting the research focus and area of contribution.

method for developing HEV powertrain models suitable for a variety of investigations such as energy management, vehicle performance and vehicle redesign or modification.

Chapter 7 discusses model management features that need to be incorporated into the management structure at both the development and systems integration level in order to implement and benefit from an object-oriented modelling method.

Finally in Chapter 8, conclusions are stated and recommendations are made for progressing this research with further work.

# Chapter 2

# HEV Powertrain Modelling

This chapter presents the differences and similarities between HEV and conventional powertrain modelling with the view of defining the requirements for a HEV powertrain design environment. Particular focus is placed on the ability to develop plant models for both lower and higher fidelity studies. For example those models required for analysing vehicle energy management or longitudinal dynamic effects. These two focus areas are selected due to the differences in the required modelling fidelity, and the differences in simulation requirements. In describing the modelling of powertrains, terminology and diagrams from the Unified Modeling Language (UML) are introduced in order to better understand the link between modelling, object-orientation and systems engineering that is explored within this Thesis. The work presented in this Thesis will show that the proposed modelling methodology is sufficiently flexible to accommodate the complexities associated with HEV powertrain design; from the variability in architectural design choices to the different levels of modelling fidelity required. Thereby assisting in achieving the ultimate objective of faster, more efficient and more consistent evolutions of HEV designs through model-based design and systems engineering techniques.

## 2.1 Complexities of HEV Powertrain Modelling

Automotive manufacturers are continuously redesigning their HEV products in order to reduce costs, improve fuel efficiency, improve on performance and include new features. This requires investigation into the many ways in which HEV power-

trains can be constructed. The aim of this section is to discuss the main reasons why HEV powertrain modelling is a more complex and extreme case of conventional vehicle modelling. Further, the highlighted differences are the reason that HEV development requires considerably more modelling effort, particularly during initial investigation and optimization design stages. The remainder of this section summarizes the differences in topologies, level of hybridization, electromechanical integration and electrical power topologies.

### 2.1.1 Multiple Engineering Domains

Hybrid vehicles make use of many more components than conventional vehicles [3]. For example, the second generation Toyota Hybrid System used in the Prius contains a combustion engine, high voltage generator and motor, power electronics, epicyclic gears and custom built battery [14]. HEV powertrains are complex systems with many configurations that combine these electronic, electrical, chemical, mechanical and control components [4, 11].

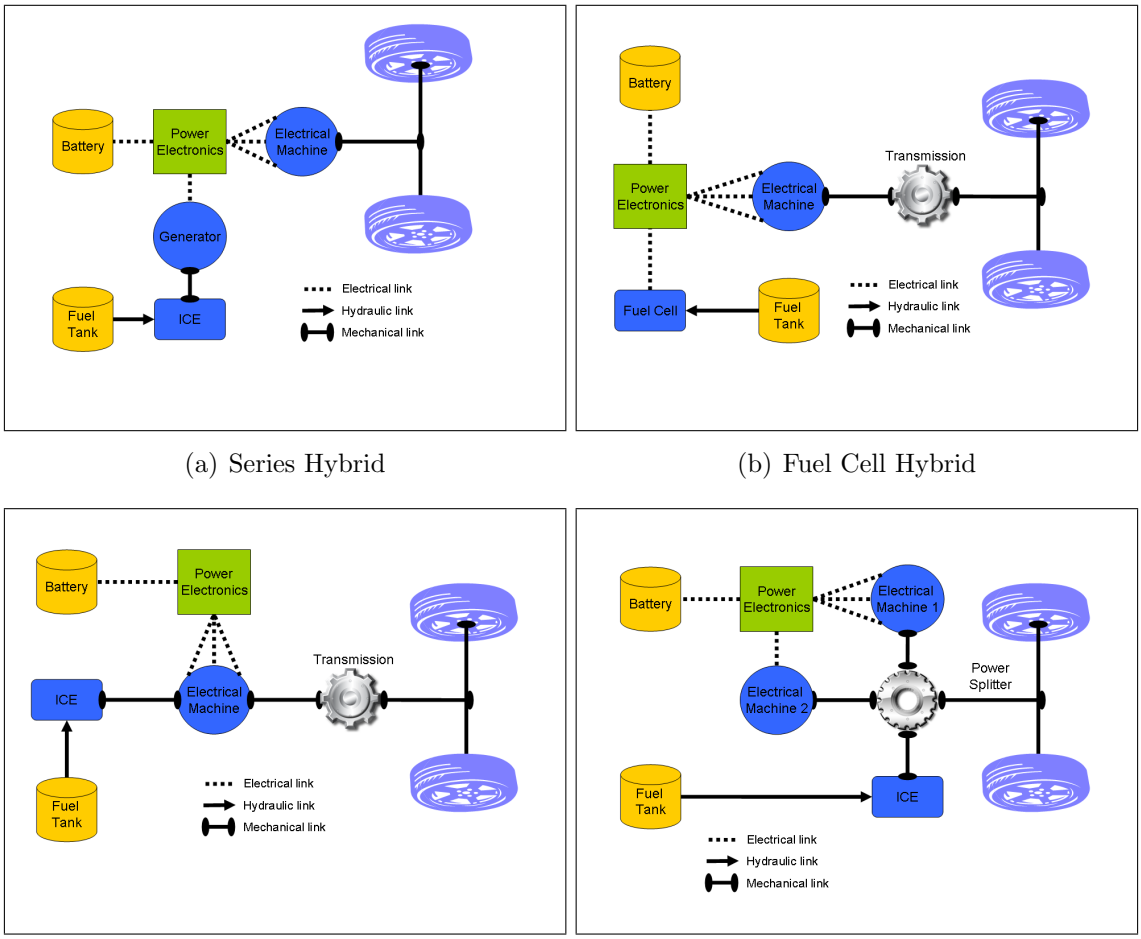
From an analytical point of view, it can be said that the design of hybrid vehicles deals with energy storage and conversion in the electrical, chemical and mechanical domains [15]. In order for a HEV powertrain modelling tool to be able to analyse the complete powertrain system there are two possible options [16]:

1. It can incorporate models from different domain specific packages via co-simulation methods. An example of this method is seen in [17] where a co-simulation approach is taken for the development of automotive control system design. Many domain specific tools exist for performing computer aided analysis and design of specific components such as the very-high-speed integrated circuit (VHSIC) hardware description language (VHDL) used for designing electronic hardware, Maxwell for electromagnetic field simulation or SimPack for kinematics and dynamics. The downside of this approach is that individual models may have to be adapted to work with the powertrain modelling tool and software patches or interfaces may be required to ensure compatibility between the different modelling tools.
2. A multi-domain modelling environment can be used which enables all components from the different engineering domains to be modelled using the same tool. Simulink, Dymola and Simplorer are all examples of multi-domain or domain-

neutral modelling environments. A possible downside is that the multi-domain environment may not provide the functionality in each case that the domain specific counterpart provides. However, it is argued that with increasing size and complexity of the system models, the benefits gained from a common tool also increase. Such benefits include reduced modelling effort and the possibility to standardize and streamline the model management as will be discussed in Chapter 7.

### 2.1.2 Vehicle Topologies

Starting from a top-down perspective, HEVs can be broadly classified into three main topological classes: series, parallel, and complex or series-parallel hybrid vehicles.




(a) Series Hybrid

(b) Fuel Cell Hybrid

(c) Parallel Hybrid

(d) Complex Series-Parallel Hybrid

Figure 2.1: Hybrid electric vehicle powertrain topologies.

**Series Hybrid**

In a typical series hybrid, two powertrains are electrically coupled together via power electronics. The driving force of the vehicle is produced via an electrical machine. An example of a series hybrid architecture can be seen in Figure 2.1(a). The main advantage of this layout is that the engine is mechanically decoupled from the wheels, allowing the engine to operate in a high efficiency speed and torque region [18]. In this region, more efficient combustion provides higher fuel efficiency and reduced emissions. The absence of a mechanical link between the internal combustion engine (ICE) and the wheels also provides the vehicle designers with more flexibility as to the positioning of the ICE and generator within the vehicle.

The main disadvantage of this layout is the double energy conversion. First the energy is converted from mechanical to electrical via the generator and then from electrical back to mechanical via the electrical machine driving the wheels. There is also additional weight and cost from the generator and the fact that the motor has to be large enough to provide all the required tractive force [19]. It can be said that the higher the continuous power demand on the vehicle, the larger the powertrain components have to be in order to sustain the performance [20].

Since a fuel cell cannot provide a mechanical force, the vehicle must be driven by the electrical machine only and the fuel cell provides electrical energy. Therefore the fuel cell vehicle (FCV) is a form of series HEV. FCVs make use of hydrogen as an energy source and a fuel cell as the converter. Compared to an ICE, hydrogen fuel cells have zero toxic emissions and have a higher energy density [21, 22]. Further, like the battery, fuel cells can directly provide the electrical machine with power without the requirement of long recharging periods [19].

Vehicles powered by fuel cells alone suffer from a low power density, long start-up time and slow power response [21, 23]. The fuel cell is not able to respond efficiently to the transient load conditions that are typical for an urban vehicle. This together with the fact that they have low efficiency at very low and very high power outputs [19, 22, 23], makes fuel cell vehicles a good candidate for hybridization. The fuel cell hybrid electric vehicle shown in Figure 2.1(b) has a fuel cell system as the primary power source, and batteries or ultracapacitors as the secondary power source also referred to as the peaking power source (PPS) [21]. Hybridization in FCVs allows for more control over how the various power demands are met and can

protect the fuel cell from peak power demands [24]. Further, since the secondary power source is used to supply peak power, the fuel cell need only be sized to supply the steady state power. This has the added benefit of size and weight reduction which in turn helps increase the vehicle efficiency [25, 26, 27]. Another advantage that comes from hybridization with a PPS, is the ability to absorb regenerative power which the fuel cell alone cannot do [22].

### Parallel Hybrid

In a parallel hybrid powertrain both the ICE and the electrical machine are mechanically connected to the wheels via a transmission as shown in Figure 2.1(c). It is possible to drive the vehicle by using the ICE alone, the electrical machine alone or by combining the use of both power sources. The power from the two sources has to be mechanically coupled, either via a torque coupling or a speed coupling mechanical device. The parallel hybrid is also able to recover braking energy through regenerative braking [28], using the electrical machine as a generator to charge the battery or ultracapacitor.

This layout has the advantages, over the series layout, of not requiring an extra generator and the electrical machine can be used as a generator when it is not required for traction. This allows the batteries to be recharged using power from the ICE, either when the vehicle is stationary or when the ICE can efficiently provide more power than required. A further advantage is that the size of the ICE and electrical machine can be reduced with respect to the series vehicle, for short-trip operation. With vehicles designed for extended travelling purposes, the ICE should be sized to meet the maximum continuous demand while the power output of the electrical machine can still be reduced [20]. The sizing of the electrical machines within a HEV is discussed further in Section 2.1.3.

### Complex (Series-Parallel) Hybrid

It is possible to combine the previous two topologies in order to form the series-parallel powertrain as shown in Figure 2.1(d). In this layout the ICE is able both to provide mechanical force directly to the wheels as in the parallel set-up, and to be used as a power source for the electrical machine as in the series configuration. The

electrical machine is able to drive the vehicle both on its own or coupled with the ICE for extra driving power. Typically this topology attempts to provide the best of both worlds [18, 28] and HEVs of this type have been shown to outperform the series and parallel counterparts in both fuel economy and driving performance [29]. Of course the performance and economy of hybrid vehicles is dependent on many external factors such as driving style and typical usage of the vehicle. It is possible that other topologies are better suited to a specific task such as short, medium to low speed travel in congested areas; both from an efficiency and design complexity viewpoint.

However, this hybrid topology also merges the disadvantages of the series and parallel HEVs. The series-parallel powertrain layout requires an additional electrical machine to be mechanically linked to the ICE in order to use the ICE as an electric power source [20]. Apart from the increased weight and cost, there are also complications with the space available for positioning the various components.

The complexity of the series-parallel HEV can be increased to allow for bidirectional power flow for both electrical machines, providing more versatility in the operation of the vehicle. HEVs with this topology demand an increase in computational complexity for managing and controlling the various subsystems in an efficient way [20]. This allows for more flexibility in the control system design where various control strategies and operating modes can be employed such as those described in [21, 30, 31, 32].

### 2.1.3 Level of Hybridization

HEVs can be further classified in terms of their degree of hybridization or in other words by the level of usage of the electrical machine as a driving power source. HEVs are generally classed as either *micro*, *mild* or *full* hybrids [20, 29]:

#### Micro Hybrid

Micro hybrids are, in essence, conventional ICE powered vehicles with small electrical machines acting as integrated starters and alternators [20]. With this class of vehicle, the electrical machine never adds power or driving force to the powertrain. This level of hybridization allows for a small energy saving, form 5% to 10% in city

driving [20], through a stop-start strategy by preventing the ICE from idling when the vehicle is stationary. Where "stop-start" refers to the ability of the HEV control system to shut down the ICE when the vehicle brakes are applied and then use the electrical machine to restart the ICE when the brakes are released. Additionally, the added weight and cost are minimized due to the small size of the electrical machine which is typically in the 3 kW range powered by a 12 V or 42 V supply.

### Mild Hybrid

Mild hybrids are also known as assist hybrids since, although the electrical machine is not able to propel the vehicle on its own, it is able to assist the ICE by adding driving power at times of increased load [19]. In most cases the electrical machine is directly coupled to the ICE and can be used as a generator for recharging the batteries either while powered by the ICE or through regenerative braking. This level of hybridization requires larger electrical machines running at higher voltages as compared with the micro hybrid. Typical electrical machines for these vehicles have power ratings in the range of 10 kW – 20 kW and nominal voltages between 100 V and 200 V [20, 29]. Mild hybrids provide a higher energy saving through increased energy recuperation, from 20% to 30% in city driving [20], but have a higher associated weight and cost.

### Full Hybrid

A HEV is considered to be a full hybrid if it is possible to drive the vehicle using either the ICE or the electrical machine. This means that both powertrains have to be large enough to power the vehicle on their own, adding considerable size and weight penalties to the vehicle design. Usually associated with a costly redesign of the conventional vehicle's powertrain [19]. Full hybrids are also differentiated from other hybrids by their large electrical machines with typical power ratings in the range of 30 kW – 50 kW, and their higher nominal voltages that are typically in the range of 200 V – 600 V [29]. This level of hybridization allows for the greatest energy saving, from 30% to 50% in city driving [20], but also the greatest weight and cost penalty.

The powertrain design of full hybrid vehicles is dependent on the purpose of the

vehicle and currently available models have taken two distinct directions [20]. Firstly there are those that make a compromise on performance in order to increase efficiency and emission reduction. In this category the size of the ICE is reduced with respect to the conventional vehicle counterpart. Secondly there are the full hybrids that are designed to improve on vehicle performance by providing extra power. Typically these vehicles are sport utility vehicles (SUVs) such as the Toyota Highlander where the ICE size is not reduced and three electrical machines are added to the powertrain.

### 2.1.4 Electromechanical Architecture

Taking a step down into the level of detail of the powertrain architecture, it is also possible to differentiate HEVs by there electromechanical architecture. This takes into consideration the number, type, placement and mechanical coupling of the electrical machines with the ICE.

#### Electrical Machine Type and Placement

Design engineers choose the electrical machine based on the properties, such as speed range, torque, power, size and cost, that best suit the vehicle's speed, range and power requirements. This choice also has a direct effect on the control techniques employed. Most HEV designs make use of one of the following three electrical machine types [19, 20]:

1. **Induction machines** – favoured as simple, robust machines with a wide speed range, developed control techniques and no back emf. Induction motors were commonly used in electric vehicle designs such as the Saturn by General Motors and the Tesla Roadster [33].
2. **Permanent magnet machines** – have higher efficiency than induction motors and can provide the same power and speed with a smaller sized motor. However, they typically have a short constant power range and large back emf at high speeds. Therefore, the power electronics used to power the machine, such as the inverter drive, must be able to withstand this back emf [20]. This technology is used in the popular Toyota Prius [34] and Honda Civic [35] hybrids.

3. **Switched reluctance machines** – have low cost, high reliability, simple control, high-speed operation with high efficiency over a wide speed range and good torque-speed characteristics. However, they are not widely produced and are difficult to design [36]. More recently, major automotive manufacturers such as FIAT, Renault and Volvo have been involved in the research and development of systems such as starter-generators and hybrid powertrains for mild HEVs using switched reluctance machines [37].

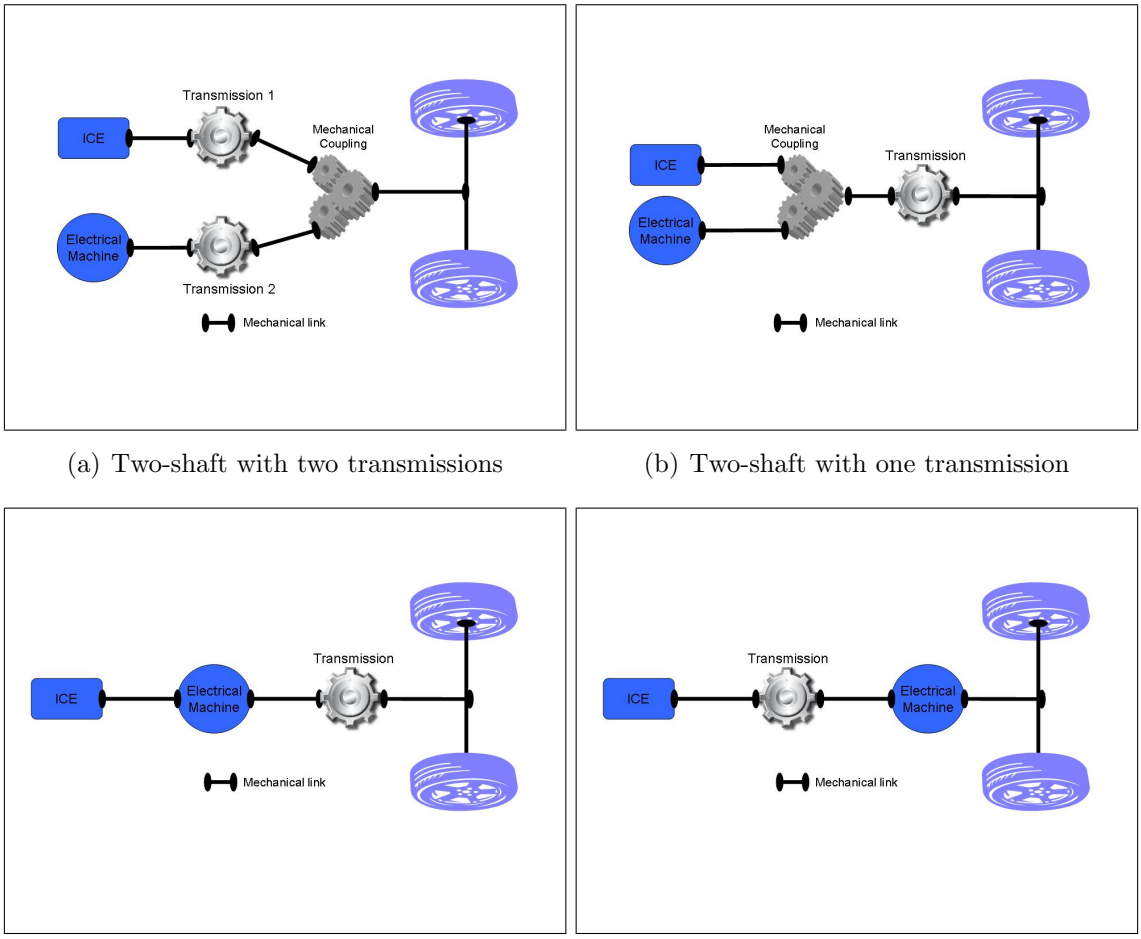
Another important choice to be made, is that of the number and positioning of the electrical machines within the powertrain. Typically there are three options available for the number of electrical machines that will provide a driving force, not including any additional non-driving electrical machines used for auxiliary purposes such as for a starter/generator. These options are listed below:

1. **One Electrical Machine** – this can either be placed on the front axle for a front wheel drive (FWD) or the rear axle for a rear wheel drive (RWD).
2. **Two Electrical Machines** – here each axle can be powered by a separate electrical machine providing four wheel drive (4WD) operation, or alternatively two wheel motors can be used for FWD or RWD.
3. **Four Electrical Machines** – this architecture allows for each wheel to be driven by an individual electrical machine. Since it is possible to individually control the speed of each wheel, no mechanical differential is required. Also, either fixed or no gearing is needed between the wheel and the motor, with the option of mounting the machines inside the wheel hub.

This architectural choice also has a direct impact on the controller design by adding additional complexities such as individual wheel speed control and increased energy management possibilities.

### Electrical Machine Coupling

With parallel and series-parallel drivetrain architectures, the defining features of the mechanical architecture are determined by the way in which the powers of the ICE and the electrical machine are combined mechanically. There are three common groupings for this mechanical coupling as listed below [19].




(a) Two-shaft with two transmissions

(b) Two-shaft with one transmission

(c) Pre-transmission single-shaft

(d) Post-transmission single-shaft

Figure 2.2: Electrical machine mechanical coupling configurations in HEV powertrains.

1. **Two-Shaft Configuration** – In this group the ICE and the electrical machine each supply their power to separate shafts which are then coupled by a torque or speed coupling device to the drive shaft. With this configuration it is also possible to have two transmissions before the coupling as in Figure 2.2(a) or one transmission after the coupling as in Figure 2.2(b).
2. **Single Shaft Configuration** – This configuration is common in parallel hybrid topologies, with the ICE and electrical machine occupying a common shaft and the rotor of the electrical machine performing the function of torque coupling. The main difference in how this configuration is implemented is the location of the transmission. The pre-transmission configuration shown in Figure 2.2(c) is used with small electrical machines such as in assist and mild hybrid type vehicles. Whereas with the post-transmission configuration shown in Figure 2.2(d), the transmission merely modifies the ICE operating point for

improved performance and efficiency while the electrical machine is directly coupled to the drive shaft. This configuration is used with larger electrical machines such as those used in full hybrid vehicles.

3. **Separate Axle** – Often referred to as "through-the-road" hybrids. With this configuration one axle of the vehicle is powered by the ICE while the other is powered by the electrical machine. This has the design benefit of conventional vehicle drivetrain on one axle and an electric vehicle drivetrain on the other, also allowing for four-wheel drive operation. A major disadvantage is the extra space taken up by the second axle powertrain. The separate axle configuration can be used for parallel full hybrid vehicles.

### 2.1.5 Electrical Power Topology

Powertrain architectures can make use of many different power sources and devices, from different types of batteries to inverters and converters. The electrical power topology considers what power devices are used and how they are interconnected.

#### Power Sources

Within a hybrid drivetrain, the primary power source provides the steady-state power, this could be a combustion engine or a fuel cell. Whereas the secondary power source or PPS provides the dynamic power, this could be an electrical machine powered by batteries and/or ultracapacitors. The way in which the different power sources are used is dependent on the relative sizing of each power source as discussed in Section 2.1.3, and the control strategy which has been employed.

Design engineers make a choice as to what type and combination of power sources should be used, considering facts such as size, weight, cost, power density and cycle life together with the overall purpose of the vehicle being designed. The choice of the particular battery type such as lead-acid, nickel based or lithium-based, and the ultracapacitor for the PPS, will directly affect the energy management strategy and how the controller implements it. For instance, an important parameter of the PPS that needs to be considered and controlled is the state of charge (SOC). Examples of modelling considerations for the power sources are seen in the Chapter 5 case study where a power supply consisting of a fuel cell in parallel with an ultracapacitor is

modelled, and then again in Chapter 6 where this supply is replaced by an integrated battery and ultarcapacitor alternative.

#### Converters and Inverters

HEVs make use of power electronic devices such as rectifiers, inverters and dc/dc converters. These need to be precisely controlled to provide the desired voltage, current and frequency, as well as bidirectional power flow [20]. The number of inverters and converters used and the way they are interconnected with the power sources and each other, such as for a floating or fixed bus voltage architecture, can vary in different HEV designs depending on space, cost and weight requirements [38].

The complexity and implementation of energy management control functions are directly affected by the chosen power architecture. For instance, it is crucial that the power balance be maintained by matching electric consumption to battery and generator output since excess demand will result in an overcurrent in the boost converter and a drop in the battery voltage. Inverters also have limits on their operating voltage range, which could be exceeded if a mismatch occurs. Tyre slip can result in large transient power imbalances and must be corrected immediately by the control system [20, 39].

### 2.1.6 Limited Experimental Data

Considering the variety of devices that can be used and the various ways in which they can be coupled into the HEV powertrain, it is clear that there are many more possibilities than in conventional powertrain design. For many of the possible HEV designs, there is little or no empirical data, prior knowledge or best practice methods available [11]. Investigating and analysing this large design space in a timely and cost efficient manner requires the effective use of modelling and simulation [3, 4, 40].

## 2.2 Modelling Concepts

In order to enable the reader to follow more easily the work presented in the following sections and chapters, a description of the significant terms used is presented here.

### 2.2.1 Model Types

Powertrain models are traditionally classified as one of the following causal types [3, 41, 42, 43]:

1. *Forward facing* – Models of this type follow a "cause-effect" calculation process in the direction of the driving vehicles physical power flow. Starting with a demand setpoint, usually in the form of an accelerator signal, torque demands for each subsystem can be calculated in a forward direction until the resultant force at the wheel is found and from this the vehicle speed. A driver model, typically consisting of a PI controller, is used to compare the desired and actual speed in order to achieve the setpoint speed. Additionally driver models can incorporate logic for steering commands, gear changes and clutch actuation.

   These models are more representative of the real world system, making use of measurable variables such as torque inputs, brake and accelerator demands. This makes them well suited to controller development and testing, in particular real-time control strategies [3]. Also it is possible to include transient subsystem models for studying the system dynamics [44]. This can include phenomena such as slip in tyre dynamics, clutch shudder and shaft resonance in driveline dynamics, pitch and roll in the vehicle body dynamics or electrical transients in power electronic subsystems [45]. The downside of this approach is that increased model complexity and bandwidth leads to smaller simulation time steps and increased overall simulation time [46].

2. *Backward facing* – These models calculate in the opposite direction to the power flow. In other words, calculations are performed backwards from the wheels to the power sources in an "effect-cause" manner. By imposing a drive cycle, tractive effort at the wheels can be calculated and continuing to move backward through each powertrain subsystem until the energy required from the input sources can be determined.

   These models are generally computationally faster than forward facing models and are therefore useful for architectural studies and optimization routines that require many iterations of relatively longer cycle times [3, 47]. Further, due to the fact that these models assume the vehicle is capable of meeting the drive cycle demands, determining the performance limits of the included subsystems is difficult. Steady-state efficiency maps and lookup tables are

generally used to account for subsystem losses and hence these models tend not to take dynamic effects into account [46].

3. *Combined backward/forward* – It is possible to combine both approaches by iterating backward simulations to achieve a target setpoint [43]. This approach requires a forward and backward model for each component. First a backwards simulation determines component efficiencies and operating limits, and then the forward facing models are used for a forwards simulation using the previously calculated efficiency and limit values [48]. The ADVISOR simulation software uses this "hybrid" approach, more details on the way this is implemented are given in [46].

   Using this type of model allows for faster simulation than the standard forward facing approach since it is possible to make use of larger time-steps and lower orders of integration [46, 49]. However, there are some drawbacks. Firstly both forward and backward facing models must be produced before any modelling activity can take place [48]. In particular the ADVISOR approach is not well suited to transient analysis and requires co-simulation with external software to perform dynamic analysis [44]. Further, designing control algorithms suitable for usage on the real-world system is made difficult by this modelling method [50]. Essentially this is due to the fact that on a time-basis, this approach is a backward facing model with forward facing loops introduced between successive systems in order to overcome the problem of performance limits mentioned previously. Therefore, the control inputs for these models don't directly translate to real-world equivalents as they do for the forward facing models.

### 2.2.2 Causality

Causality in modelling refers to the cause and effect structure of a model and its components, from inputs to outputs. A model is said to be causal if the relationship between the inputs and outputs are described in a fixed sequential manner. Causal modelling, more commonly referred to as block-oriented modelling, requires the modeller to determine the causality of the model and describe the relationship between the inputs and outputs in terms of ordinary differential equations (ODEs).

A popular environment that makes use of causal block-oriented modelling is the Sim-

ulink package. Simulation packages such as Simulink require the user to rearrange equations depending on the causality of the system being modelled, meaning that separate models need to be maintained for each possible causality. This includes different models for the forward and reverse dynamics of the same component.

Non-causal models, also called acausal, do not specify the direction of calculation from input to output but describe this relation in the form of a differential algebraic equation (DAE). In this way the modeller only has to describe the physical laws for each component and the translator within the simulation environment will determine the causality of each component and the overall model. This is done by examining how each component is connected to the next one along with which variables are known and which need to be determined, thereby stipulating inputs and outputs for the final model.

Modelica is a physical modelling language, as defined in Section 3.1.2, that allows for the creation of non-causal component models which can be used in both forward and backward facing applications [51]. In other words, each component model can be used in both a forward and a backward fashion by merely changing how the inputs are defined [3]. This is because components are modelled without regard as to the form of the equations, meaning that DAEs can be used in the model description; or the direction of cause-effect relationships, otherwise termed causality. In this case, the causality of the underlying component equations is determined during the optimization and translation of the Modelica source code. It is the task of the translator within the simulation environment to do this by examining the model structure for inputs and outputs, and algebraically sorting the equations through the process described in Section 2.2.4.

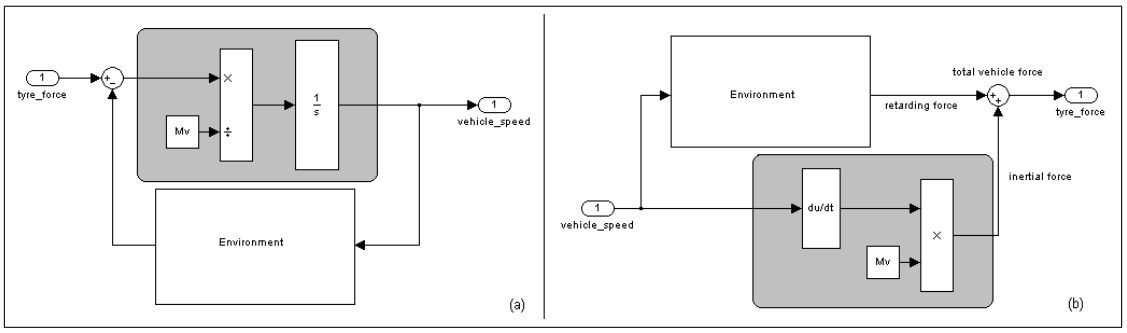



Figure 2.3: Causal vehicle mass models:(a) forward dynamics (b) reverse dynamics.

### Examples of Causal and Non-causal Modelling

A typical example of block oriented causal modelling is shown in Figure 2.3, where two vehicle mass models required for forward and reverse dynamics are presented. The highlighted areas in Figure 2.3 show how two different models must be built in order to implement the force equation for the two different causalities. In (a) the integral of $F/m$ is solved to find the vehicle speed and in (b) the product $m.dv/dt$ is solved to find the required tyre force.

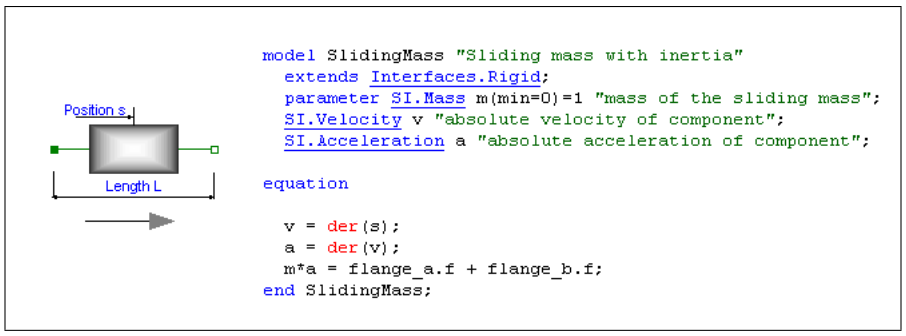
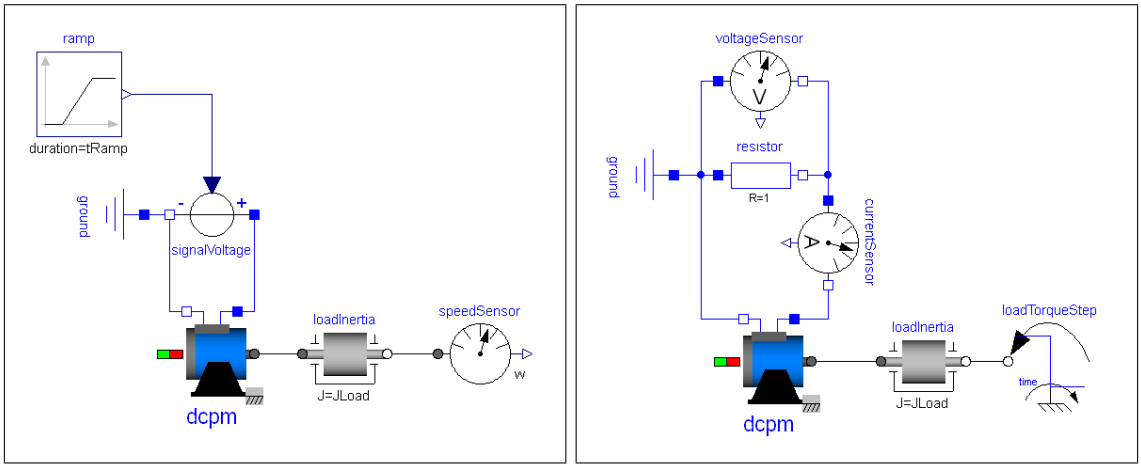



Figure 2.4: Description of mass component in non-causal language.

With an equation oriented language, a mass component can be defined by the relation $F = ma$ as shown in Figure 2.4. The implementation defines a mass with two flanges attached to it, a driving flange (`flange_a`) and a driven flange (`flange_b`), where $s$ is defined as the position at the centre of these two flanges. The resultant force on the mass is then defined as the sum of the forces at each flange (`flange_a.f + flange_b.f`) since the positive direction is used for both flanges. This same component can then be used in any model regardless of the causality.

Another example of how non-causal models can be used to model different causalities is shown in Figures 2.5(a) and 2.5(b). In this example the non-causal plant is represented by a simple DC motor model connected to a load inertia. Figure 2.5(a) shows the DC motor model being supplied by a voltage ramp in order to determine the resulting speed of the load in a forward facing manner. In Figure 2.5(b) a torque is applied to the load so that the voltage and current at the input of the DC motor can be calculated in a backward facing manner.

Alternatively, Figure 2.6 shows another method for modelling the reverse causality of the model when using languages such as Modelica. In this case the model is inverted by setting the speed output to a given speed-time profile and letting the




(a) Voltage fed DC motor model (b) Torque fed DC motor model

Figure 2.5: Non-causal usage of DC motor model.

voltage input be the new output. The inverse plant model is calculated by the code translator in order to calculate the power required to follow the given speed profile. In order to use this method the model developer must make sure that the original plant model is invertible. In other words, it may be necessary to make some modifications to a complex plant model in order to avoid inversion errors such as inverting table data or infinite inverse solutions [52]. The implementation of invertible models was not considered in this Thesis as this is mostly applicable to the development of controllers based on inverse dynamic models. The interested reader is referred to the literature for further discussion [52, 53, 54].

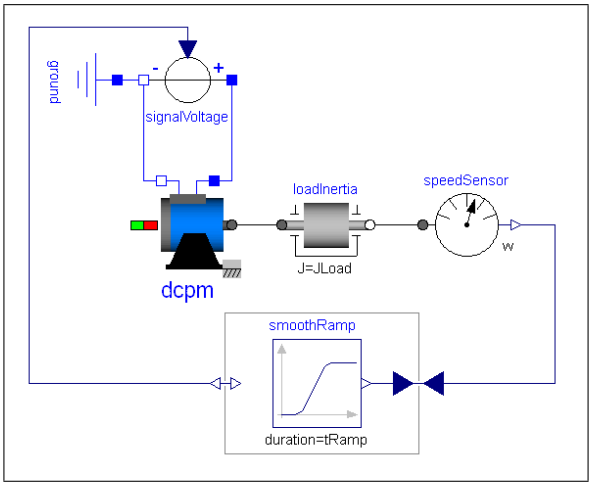



Figure 2.6: Inverse DC motor model to for determining reverse dynamics.

In all three of these model examples, only one non-causal implementation of the DC

motor and inertia has been developed. The same plant model can be reused in each case, since it is the task of the code translator and not the developer to arrange the model equations to suit the selected causality.

### 2.2.3 Model Fidelity and Stiffness

In basic terms fidelity can be described as a measure of how accurately a model describes the properties and behaviour of the real object being modelled. Where a model with the highest fidelity has the closest approximation to the real world object. Another way of referring to fidelity from a programmatic or model building point of view, is to refer to the logical complexity of the model.

Logical complexity of models is determined by the amount of detail that has been incorporated into each component within the model. The target requirements of the simulation dictate the level of detail needed in the components used. For instance: low-fidelity models for feasibility studies are usually map-based, medium-fidelity models for control strategy assessment typically make use of physical models for describing important effects while empirical data can describe less critical parts, and high-fidelity models for component design generally include detailed descriptions of the physics concerned [2, 55]. For example, motor torque investigations can make use of idealized voltage sources for motor control since simplified models reduce runtime, whereas for transient investigations of the effects of power electronic switching, more detailed power electronic and controller models are necessary [2].Two commonly used divisions in modelling fidelity are those of steady-state and dynamic models.

Steady-state models have the advantage of being faster to compute and can therefore be used for longer simulations. Typically useful for evaluating high-level operating strategies and architectural options of a design [3, 56].

Dynamic models incorporate a much higher level of detail through physics-based equations describing time-varying attributes such as electrochemical and thermal reactions within a battery. For example a steady-state model of a shaft can be a rigid inertia whereas a dynamic model would include flexibility to enable the modelling of shuffle effects within a driveline [57]. These models are used for detailed subsystem analysis and design. More specifically, they are useful for determining the requirements of power electronic devices within the HEV powertrain, the analysis of

noise, vibration and harshness (NVH), and the analysis of vehicle handling [3, 56].

If a high level of detail is required from the simulation results, more time is required to construct, run and analyse the results of high-fidelity models [3, 56]. In other words, the main disadvantages of using high-fidelity models are: the increased development time; difficulty in parameterizing the model due to the many parameters needed to accurately describe devices, such as the stiffness and damping of shafts; and the greater computation time required due to the increased number of equations [56].

Another issue with the use of complex dynamic models containing many differential equations is the increased likelihood of model stiffness. The model is said to be stiff when it has both fast and slow varying dynamics in its solution [58]. Where the stiffness of the model or system refers to phenomenon presented by the system, rather than a specific property of the system, since there is no precise definition of system stiffness [59]. This is often caused by different time scales in the dynamics, for example when combining electrical systems with typical time constants in the 1–10 k$Hz$ range with mechanical systems having time constants in the 10–100 $Hz$ range or thermal systems with time constants less than 1 $Hz$.

In order to deal with increased model stiffness and avoid incorrect results due to numerical instability, appropriate numerical integration algorithms must be selected when simulating the model. Stiff models usually require variable-step numerical integration methods that are capable of reducing the step size for the fast dynamics during the simulation, since reducing the overall step size increases the computational load. The topic of numerical techniques is discussed further in Section 3.3.3.

### 2.2.4 Symbolic Manipulation

Symbolic manipulation or computation refers to the ability of a code translator and optimizer to take the system of initial equations set up in the model and solve them in an algebraic manner rather than a numerical manner. A simulation front-end for the Modelica language, such as Dymola, automatically rearranges the physical equations describing each component via symbolic manipulation, deciding inputs and outputs from the context in which each component is used. This manipulation of the model equations is necessary in order to simplify the final system of equations

needed for simulation into a form that is efficient for numerical analysis [60].

Since non-causal languages like Modelica allow for complex multi-domain models of multiple components described by DAEs, this can lead to several hundred thousands of equations. This feature is very useful in object-oriented modelling since it allows a modeller to develop at a high level of abstraction by intuitively connecting components and subsystems [61]. However, when object-oriented code is flattened by removing the object-oriented structure and replacing inherited classes by their complete code, the final code generated is much larger than the original model and can no longer be intuitively correlated with the original system being modelled. With equation-oriented models, this means that the flattened model contains many sparse equations, of which many are repeated and some have high order derivatives.

Computer algebra techniques aid in reducing computation time and the possibility of errors due to numerical differentiation [62]. Symbolic manipulation both reduces the number of equations to be solved and the DAE index to 1 or 0 [58]. The DAE index refers to the minimum number of differentiations required in order to achieve an equivalent set of explicit ODEs [12]. This index reduction is done by symbolically differentiating certain equations and substituting the derivatives back into the original system of equations [58, 63]. Further, symbolic manipulation can eliminate systems of simultaneous equations through substitution, thereby removing algebraic loops and reducing simulation time [63]. The final code is optimized for numerical integration to a level that would be a significant challenge for a developer to derive from first principles [61], as would be necessary in a causal block oriented environment.

Fritzson explains that the occurrence of high index equations is typical in mechanical and mechatronic models due to the constraints between connected models, and that this is a property of the object-oriented modelling method and not the system being modelled. Therefore, the ability to automatically transform such equations through symbolic manipulation is indispensable in order to allow for the effective object-oriented use of reusable component models. For example, Tiller et al. [64] develop a detailed vehicle powertrain model using the Modelica language which consisted of approximately 250 000 equations before manipulation and 25 000 after the reduction process.

## 2.3 Generic Powertrain Modelling Features

Many aspects of HEV powertrain modelling can be thought of as generic and therefore a necessary feature in any powertrain modelling process, whether relating to a hybrid vehicle or a conventional powertrain using an ICE.

### 2.3.1 Subsystem Modelling

If a multi-domain modelling tool is used, subsystem models can be constructed to represent the counterpart subsystems of the real world powertrain. Most of the subsystems used in a conventional powertrain design and their constituent components are also needed in HEV powertrain design. These include the wheel and tyre models, brake models, chassis models and engine models. Additionally the following subsystem models can be reused but may need some adapting for HEV purposes:

- driveline models – which may need changing from the conventional vehicle driveline models due to the possible unconventional mechanical architectures in HEV powertrains such as wheel motors or separately powered axels,
- transmission models – depending on the chosen hybrid configuration a transmission model may not be required or it may need to be changed or modified in order to couple the mechanical and electrical torque systems,
- driver models – depending on the complexity of the driver model these may have to be adapted to the specific needs of the hybrid vehicle but are generally designed for the simulation test being performed.

Gao et al. [3] state that good modelling environments provide large varieties of vehicle components for the flexible construction of vehicle models by users. Therefore the ability of a tool to allow users to share and reuse previously built and tested models and components by means of libraries can save considerable development time and effort [9].

### 2.3.2 Connecting Subsystem

With physical modelling tools it is possible to apply the principle of conservation of energy for dealing with the connection of subsystems. In other words, for a closed

system, all energy flowing into a specific point should sum to zero if no storage is assumed [7]. This can be applied to any domain provided that appropriate choices are made for the variables representing the energy carrier. These variables are often referred to as either flow or through variables.

Additionally a variable representing the amount of energy at a point, referred to as either the potential or the across variable, should also be defined. At a specific connection all potential variables should be equal. Example potential and flow variables for different domains are given in Table 2.1 from [7].

Table 2.1: Potential and flow variables for different physical domains.

| **Physical Domain** | **Potential Variable** | **Flow Variable** |
|---|---|---|
| Electrical | Voltage | Current |
| Mechanical (translational) | Position | Force |
| Mechanical (rotational) | Angle | Torque |
| Hydraulic | Pressure | Volume flow rate |
| Heat | Temperature | Heat flow rate |

A direct benefit of using this method to define connectors for each type of domain model, is that of acausal modelling, since the conservation law at the connector must be upheld irrespective of the model causality.

### 2.3.3 Simulation

To gain the most benefit from the developed powertrain models, they should be capable of both off-line and real-time simulation. Particularly if a model-based development process is being employed, simulation of the physical models forms an important part in analysis, optimization and calibration of new powertrain designs [7]. In order to test the feasibility and predict the performance of any particular design, off-line simulations are necessary. This type of simulation is also useful for developing initial control strategies. To further optimize and calibrate system variables and controllers, real-time simulations are required.

The modelling tool must either contain the necessary numerical techniques to perform simulations, as is discussed in Section 3.3.3, or it should be able to convert

the model into a format that can be used by other analysis and simulation tools as described previously in Section 2.1.1.

## 2.4 HEV Powertrain Modelling Requirements

### 2.4.1 Conventional Powertrains Design

Conventional powertrains can almost be thought of as a subset of hybrid powertrains, since many HEVs, such as micro and mild hybrids, are primarily modified conventional vehicles. Using terminology from the systems modelling languages, this relationship can be described as the conventional being a *generalization* of the hybrid. Alternatively, another way of describing this relationship would be to say that HEV powertrains are a *specialized* version of conventional powertrains. Of course this is a simplification as it is not always as simple as merely adding an extra component without making changes to some of the existing components such as the control algorithms employed. However, it is arguable that many of the components and features remain unchanged and the general principle that the HEV powertrain requires additional features is correct. This general concept of specialization is illustrated by the UML class diagram in Figure 2.7 which shows the inheritance relationship between a hybrid and conventional class.

### 2.4.2 Electrical Subsystems

Clearly one of the main differences between conventional and HEV powertrains is the major presence of electrical components such as batteries, ultracapacitors, inverters, converters and electrical machines. For example, Schupbach and Balda [65] compare three DC-DC converter topologies for interfacing with an ultracapacitor in a HEV power management system. Burke [66] reviews different battery and ultracapacitor technologies for use as energy storage in different types of HEVs and EVs.

HEV modelling requires that these electrical subsystems be adequately represented within the powertrain model through an equivalent electric circuit model or other representative model such as a lookup table or black-box model. It may also be necessary to model any device specific controllers in order to correctly simulate the

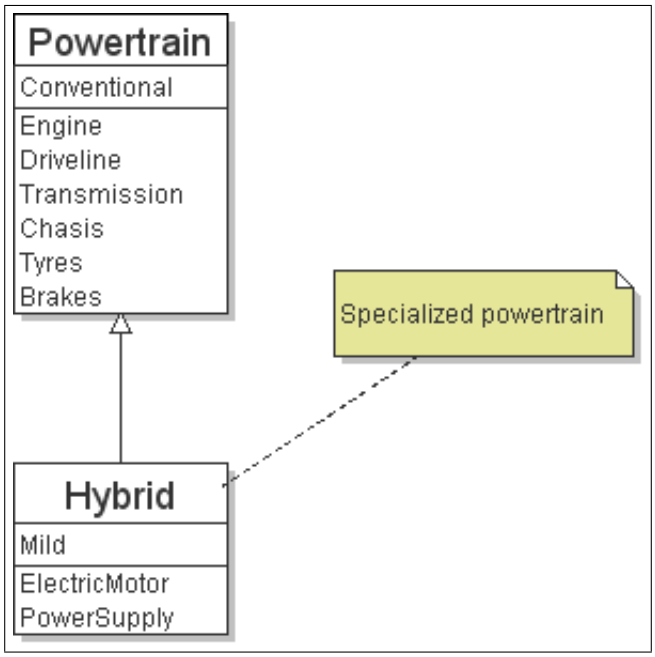



Figure 2.7: Relationship diagram.

functionality of the device as illustrated in Figure 2.8.

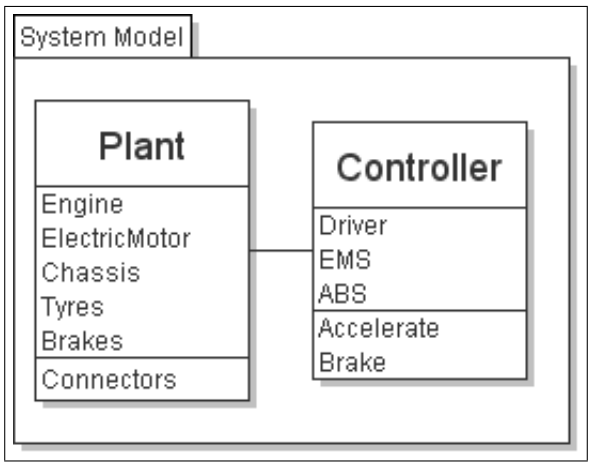



Figure 2.8: System consisting of plant and controller classes.

### 2.4.3 Architectural Studies

Many of the available simulation tools, such as ADVISOR, SIMPLEV and CarSim make use of fixed powertrain architectures [49]. One of the main purposes for the HEV modelling environment will be to investigate the possible advantages offered by the many different architectural configurations that are possible with a hybrid

vehicle powertrain. Therefore, a key requirement for HEV modelling is flexibility in architectural design. Also the ability to perform trade-off and preliminary optimization studies is important in making design decisions early in the design life-cycle.

### 2.4.4 Modelling and Simulation Based Analyses

A large part of the simulation and modelling activities in the automotive industry is for the purpose of developing, testing, optimizing and calibrating the various vehicle system controllers. For this work a broad distinction is made between the lower fidelity and higher fidelity models. Where lower fidelity models are typically used for initial investigations of long term effects such as vehicle fuel consumption or the development of energy management control strategies. The higher fidelity models are typically used for more detailed investigations of the vehicle dynamics and faster transient effects and allow for more precise controller calibration.

#### Energy Management

Energy management can be described as the task of controlling how the power is split or distributed between the various sources during vehicle operation [67]. It is the precise control of the energy flow within the powertrain that allows the HEV to realise its optimum potential in terms of fuel economy and efficiency [67, 68]. This is typically achieved by implementing an energy management control strategy.

In a conventional vehicle or pure electric vehicle the energy flow between the subsystems occurs in a fairly fixed fashion. Hybrid vehicles by definition must have more than one power source and therefore require a system for determining how and when each source gets used. In this case the power distribution does not occur in a physically predetermined manner [69], in other words, the energy management strategy must be taken into account when simulating the performance of HEVs. Further, since there are many possibilities for HEV designs there are also many possibilities for energy management strategies.

Typically, specific powertrain models are used in energy management studies in order to predict fuel usage and emissions and to develop and optimize the energy management control strategies. For example, Koot et al. [70] investigate several control strategies for managing the use of a power controlled alternator to influence

the engine's operating point. For this study the authors make use of a power-based model where the mechanical output power of the engine is split between the propulsion power and the alternator power, which is in turn split between the battery and electric loads. Static map-based models are used to describe both the engine, which maps the fuel rate to engine power for varying engine speeds; and the alternator, relating input and output powers to the engine speed. The battery model includes a quadratic loss function and represents the SOC as the integral of the power over the capacity. Several control strategies were investigated by simulating this model over the NEDC drive cycle having a duration of 1200s.

Ceraolo et al. [71] consider a generalized method for HEV energy management. They too take a power based approach to constructing their models, where the propulsion power is made up of power from a fuel converter which could be an engine or fuel cell with electric motor, and power from an energy storage device such as a battery or an ultracapacitor. The role of the energy management system in this case is to use the driver inputs to determine how to proportion the propulsion power between the available sources. Interaction between the plant model and the controller occurs in three main areas:

1. Fuel converter – on/off signal for fuel or engine start/stop,
2. Energy storage device – energy level monitoring through SOC,
3. Electric motor – torque request.

Developing the energy management control system requires numerous simulations of a sufficiently long duration, in the order of several minutes, to allow for plant and controller parametrization. In [38] a low bandwidth model of the high voltage power system for a HEV is used for this purpose, since the dynamics of the high bandwidth model do not allow for sufficiently fast simulations. Simulations for energy management analysis must consider all the vehicle operating modes, subsystem efficiencies and the overall performance of the powertrain over one or more complete drive cycles. Sample North American, European and Japanese drive cycles are shown in Figure 2.9. Since long term effects such as fuel economy and SOC are more important in this type of analysis, the type of plant model required for these simulations does not need to take into account the transient dynamics that are necessary for higher fidelity investigations such as driveability. A lower fidelity quasi-static plant

model can therefore be used in this case since the transient effects such as switching and vibrations are much faster than the energy flow dynamics [67]. Typical outputs for this type of study are; the fuel flow rate in grams or litres per kilometre, and the change in SOC of the energy storage device, for a test cycle with a duration in the order of several minutes to an hour.

It is clear that from a modelling perspective, the types of models used in energy management studies must be capable of fairly fast simulation since typically these investigations will have to cover all of the intended driving states and conditions that the vehicle is being designed for.

### Higher Fidelity Simulations

An example of modelling and simulation studies requiring higher fidelity is that of driveability studies. Driveability is generally concerned with the longitudinal dynamic behaviour of the vehicle in response to various driving conditions. Though much of the evaluation of driveability is subjective and commonly performed by test-driving prototype vehicles, objective evaluation criteria are necessary if this evaluation is to be performed via modelling and simulation. In an effort to isolate some objective criteria for some of the more subjective factors perceived to affect vehicle driveability, Cacciatori [47] proposes the following three physical variables as dominating the driveability perception:

1. Delay time – the delay between pedal push and acceleration change.
2. Acceleration value – the peak value of the initial acceleration phase.
3. Jerk value – first acceleration divided by its duration.

It is important to realise that driveability is not only determined from the perceived or psychological expectations of the driver, but also from direct physiological effects that may cause annoyance or discomfort. One of the main causes of this discomfort is the existence of low frequency noise and vibrations [72]. Therefore it is important to consider the low-frequency behaviour of the powertrain in order to be able to assess and control unwanted noise and oscillations. In particular the frequency range between 0.5 $Hz$ and 80 $Hz$ contributes significantly to agitating the human body [72].

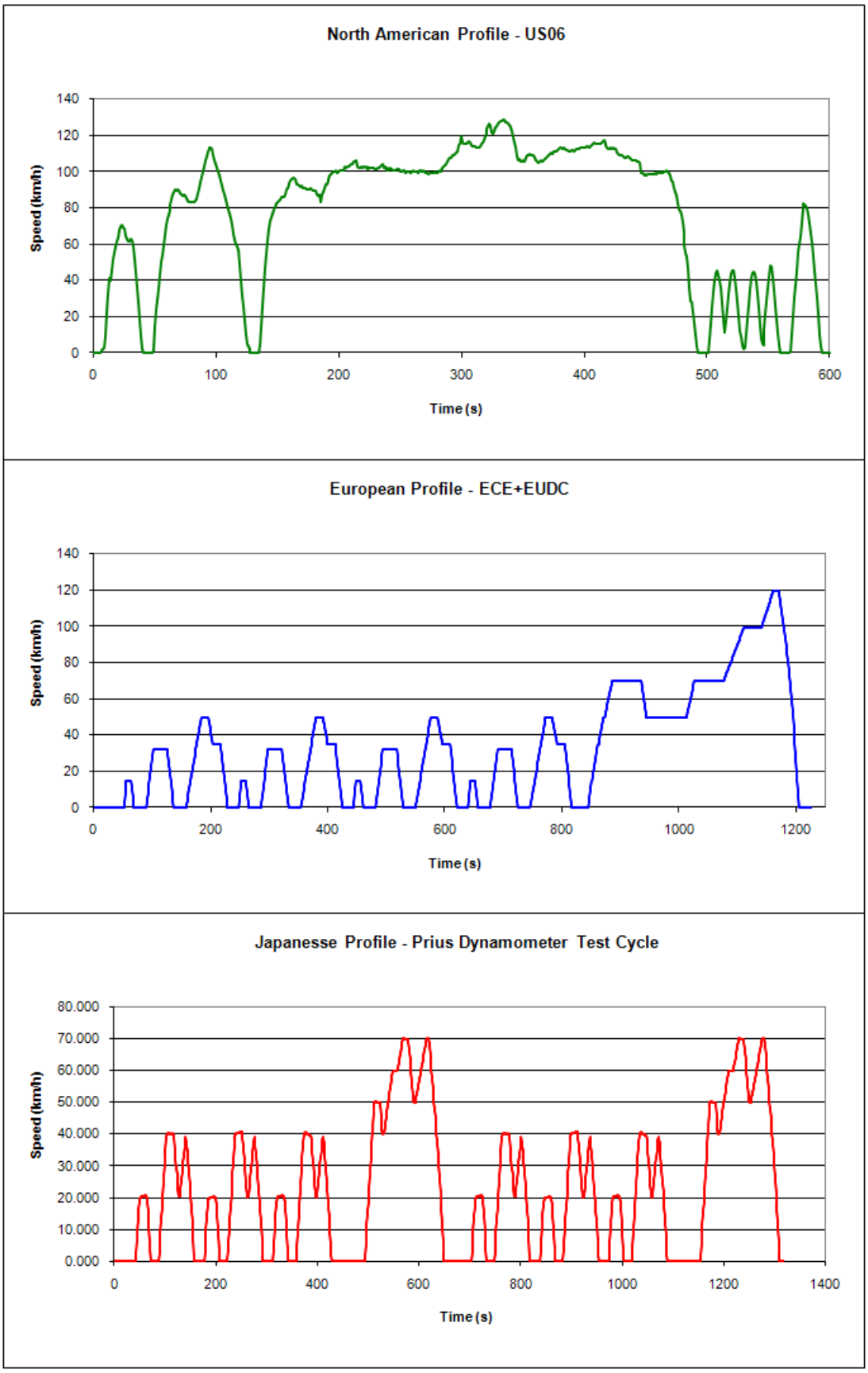



Figure 2.9: Sample drive cycle speed profiles used in USA, Europe and Japan

Grotjahn et al. [73] look at modelling driveline dynamics in order to design an "anti-jerk" controller. To investigate the effect of large driveline torques on driveability, the authors produce two lumped parameter driveline models. The first is a model of two inertias linked by a spring and damper. The engine's viscous friction is also considered when defining their equations of motion. In the second model a three-mass system is used to represent vehicles with a dual mass flywheel. These models are then used with both off-line and on-line optimization methods in order to determine the required controller parameters. In this study, a "tip-in" and "tip-out" test is performed by stepping the engine torque from $0 - 150\,\mathrm{Nm}$ and then back to zero again.

Further, Grotjahn et al. [73] show how dynamic model-based simulations can be employed to reduce the time/cost of the "anti-jerk" controller parametrization process. In this study two performance constraints are used for determining the range of controller parameters, being driver demand for sporty behaviour and comfort. The authors highlight that these two aims are in conflict since sportiness requires sharp acceleration whereas minimal oscillations are required for comfort. Their results show regions of parameter combinations constrained by sportiness and comfort, with the simulated optimum having negligible difference from the expert tuned solution. Noting that these simulations should be used to reduce the set of parameters that require testing to a smaller set or region of the most suitable parameters. Following this, faster fine-tuning by an expert in order to achieve the desired subjective performance is possible.

Fredriksson et al. [74] research different control methods to actively damp driveline oscillations due to the elasticity of components such as driveshafts. Accelerating these components with high torques can cause low frequency resonant vibrations known as shuffle. In their study they show that the resonant frequency of the driveline model is dependant on the gear ratio. Higher gears having higher resonant frequencies but lower magnitude peaks. These frequencies affect driveability when induced and lie in the range of 0–40 $Hz$.

To perform this study Fredriksson et al. [74] make use of a third order model comprising of transmission, flexible driveshafts, wheels and chassis. The author's state that this model sufficiently represents the driveline with the assumption that the driveshafts are the main cause of oscillations since they experience the largest torque with respect to stiffness. Also a stiff clutch and propshaft is assumed, with the main

drawback of this model being that it does not account for tyre slip. They conclude that better driveability prediction requires the ability to simulate complete vehicle dynamics.

Dempsey et al. [75] study the effect of tip-in and tip-out manoeuvres on the vehicle acceleration. For this study the powertrain model includes: transient engine torques; transmission, driveline, tyre-slip and chassis interaction; and an engine control system. An existing mean-value engine model and controller from a fuel economy study was reused and coupled to a lumped parameter transmission model with inertia, damping, stiffness and backlash referred to the input shaft.

Different levels of driveline and chassis models were used in [75] to compare simulation accuracy and performance. For the basic driveline model, each shaft has two inertias joined by a spring-damper and the basic chassis includes the vehicle body, suspension and tyre slip models. In both cases more complex models included a multibody differential and non-linear mount between the chassis and differential. An "extreme" model was also built using multi-element models for the propshafts and halfshafts in order to represent the joints and couplings. A comparison of results for a tip-in test shows that the "extreme" and complex models both closely match the test results but the simulation time is over 22 minutes in the "extreme" case versus 8 minutes for the complex case. The complex model takes almost 4 times longer than the simpler models which produce a comparable level of accuracy for conducting sensitivity studies.

As far as the powertrain modelling is concerned, driveability studies can have several implications on the construction and usage of the model. For instance:

1. Manoeuvre based scenarios replicate driver demand inputs rather than drive cycles. Typically with a duration in the order of seconds, these simulations are performed for particular situations that occur during driving such as tip-in, tip-out, brake to zero speed and fast gear changes.

2. The fidelity of component models must sufficiently represent the phenomena being investigated such as driveline shuffle and gearbox rattle. Some typical examples of how bandwidth is added to the powertrain model are the inclusion of low inertias, torsional compliance and non-linear effects including viscous damping.

3. These physical variables must be assessed under the different operating modes or states (such as starting, accelerating, decelerating, regenerative braking and recharging) of the vehicle in order to adequately calibrate a controller for driveability. Therefore the model should be suitable for optimization and parametrization studies.

Since the normal practice for assessing driveability is by means of experimental trials conducted by an expert, requiring time-consuming test drives, having the ability to perform simulations with different levels of models can help in narrowing down the possible parameter values and thereby aid the engineer in reducing the number of trials required to fine tune optimal performance [73].

### 2.4.5 Implications for the Modelling Environment

Considering the diverse topics covered in this section, a list of several requirements for a HEV modelling and simulation tool can be generated. The following list summarises some of the pertinent features which will enable a user to investigate multiple HEV powertrain designs using models with a flexible and reusable structure. Two categories have been defined, namely generic features which can be considered as useful for any modern modelling environment and object-oriented features which are considered necessary for implementing an efficient model-based design methodology.

#### Generic Features

1. Having established that HEV design incorporates a combination of mechanical systems, electrical systems, controllers and possibly other systems, it stands to reason that in order to effectively study HEV powertrains the environment used should be capable of multi-domain modelling.

2. Storage of previously built and tested models, subsystems and components is also necessary. If possible these should be stored in libraries that make them available to other users of the environment in a consistent and well managed manner as described in Section 7.1.1.

3. From a simulation point of view, the tool must be able to compile the model into a mathematical form that can be simulated. The simulator must support

various numerical integration techniques for solving the system of equations described by the model. Additionally both fixed and variable step solvers should be available for off-line and real-time simulation purposes.

4. Since modern powertrain designs are dependent on embedded control software, the environment should make provisions for developing and evaluating controller functions.

5. A means of graphically representing and interrogating all simulation results and variables should also be provided. This is particularly useful during initial component design and testing stages as it removes the need to export results to an external analysis environment.

6. Different types of users are likely to make use of the modelling and simulation tool. Typical users may have different specialities such as component or system designers, control designers and computer aided design (CAD) technicians, each having a different level of knowledge with respect to the functionality of the overall model. The modelling and simulation tool should therefore be intuitive to the extent that it can be used by all levels of users and at the same time provide sufficient flexibility to expert users to perform detailed work in their domain. This can both promote sharing of knowledge and work between different user groups within an organization, and reduce the need for redundant modelling. Further, this must be supported by management decisions relating to documentation, model sharing and communication as discussed in Chapter 7.

**Object-Oriented Features**

1. Equation oriented modelling is necessary in order to provide the non-causal properties required for object-oriented design. Component and subsystem functionality should be described mathematically and where possible this description should be as close as possible to the physical laws that define its operation. The ability to offer a non-causal model description also promotes the reuse of models in different applications, thereby reducing modelling time and effort. In order to allow for non-causal equation oriented modelling, it is necessary for the modelling environment to include a code translator that

can symbolically manipulate the model equations for numerical integration in order to produce an optimized simulation code.

2. A key feature for object-oriented modelling is a topological interface that allows for model structures to more closely represent the real-world systems being modelled. Together with the use of component libraries, this makes the modelling process more intuitive and provides a simple method for visually modifying model architectures. This feature enables the user to model large complex systems as a series of interconnected subsystems at different hierarchical levels. Within the lower levels of the hierarchy the subsystems are described by interconnected components. The importance of the model library structure and availability of a means to navigate through the system model at different levels is elaborated in Chapter 7 Model Management.

3. It is important to be able to clearly and accurately describe how these components and subsystems can interface with one another. In other words, connections must be defined for data transfer between the components used to make up a model. If possible a connector should be defined to represent each type of real-world connection used. For example connectors to represent data signals, electrical connections, mechanical couplings, thermal coupling, etc. Since each model or object definition must be independent of whatever other object it may be connected to, the connector definition must itself provide equations to ensure continuity at the connection point.

4. Since the tool is to be used by multiple users with different needs, it can be assumed that there will be a need for different levels of model fidelity. In order to reduce modelling effort, it should be possible to replace components and subsystems within a model with higher or lower fidelity versions of those subsystems. This feature is closely linked with the specification and enforcement of a specific library structure as is discussed in Section 7.1.1. Further, it is also of importance for computer aided engineering as described in Section 7.2.2, for the purposes of software integration and automation.

It is noteworthy that there are many modelling tools and software packages available on the market that are used in industry applications and meet the needs of a specialized domain such as electronic design and finite element method tools for mechanical and electromagnetic design. However, no tool is capable of the entire

design life-cycle of a product in an integrated manner [76, 77]. King et al. [1] point out that in this situation the best-of-class tools can be used and then integrated to form a complete system. In practice it would seem that it is precisely this integration step that forms the bottleneck in integrating an overall computer aided engineering system, either due to incompatibility issues or lack of integration software.

The motivating factors for HEV powertrain modelling as referred to in this research are mainly the reduction of time and cost through:

1. model-based analysis to increase confidence and reduce prototyping,
2. reduced modelling effort through the reuse of models both within a design process and for new designs, and
3. management of complex design models using object-oriented and systems engineering principles.

For this purpose the required tool does not have to support the entire design life-cycle but should be capable of supporting the initial concept design and refinement stages. Sufficient modelling accuracy is needed, in each of the represented domains, to perform off-line and real-time hardware-in-the-loop (HIL) tests.

# Chapter 3

# Object-Oriented Modelling and Simulation

In the software engineering domain, "objected-oriented" is an adjective used to describe a programming language that can support a style of programming whereby the program is described by self-sufficient code modules or *objects*, which can both process data and interact with other objects. Object-oriented programming also makes use of some fundamental concepts such as encapsulation, modularity, inheritance and abstraction, which allow for easier programming [8, 78, 79]. In order to better understand how these concepts are useful in the development of simulation models, definitions for some of the important terminology used in this field are provided for reference in Appendix A.1.

## 3.1 History of Object-Oriented Modelling

Modelling and simulation is not new to engineering. Analogue simulators prevailed between the 1920s and the 1950s. Systems were modelled using ordinary differential equations (ODEs) and then mechanical devices were constructed that simulated the equation characteristics. This was achieved using devices such as gear boxes, cams, ball-and-disc integrators and torque amplifiers. Towards the end of the 1940s it was shown that electronic devices could be used to perform these analogue simulations, using potentiometers, operational amplifiers and voltages instead of angles to represent system variables. This had the effect of commercialising the use of

analogue computing and the associated paradigms and methodologies became more widespread [80].

Various changes occurred in the field over the years, driven both by user needs and advances in techniques and technologies such as a move from mechanical analogue devices to electronic analogue devices. Another step change occurred with the introduction of digital computers in the 1960s and then again in the 1990s with the advances in personal computing. One particular example is the operational amplifier which underwent a technology transition from vacuum tube, to solid state device, to integrated circuit in this time period. This section highlights why the more widespread modelling tools follow a block-oriented approach and the thinking that leads to the development of non-causal physical modelling tools and languages, such as Dymola and Modelica.

### 3.1.1 Analogue Paradigm

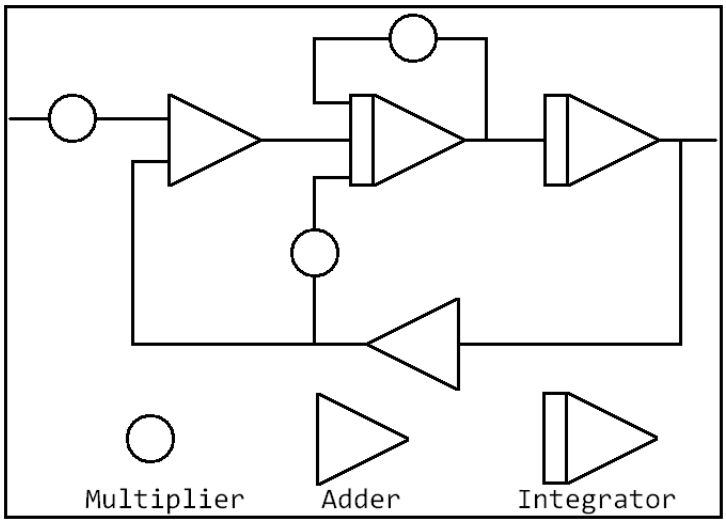



Figure 3.1: Sample form of an analogue simulation diagram.

Analogue simulation techniques required the system differential equations to be represented in terms of addition, multiplication, integration and function generation [80]. To do this, suitable state variables are chosen, usually the differentiated variables, and the system equations are manually transformed into a state space form. Once in this format, an analogue simulation schematic of the form shown in Figure 3.1 can be produced using only adders, multipliers and integrators. This process is necessary, since trying to simulate the system equations in their basic physical law

form as differential algebraic equations (DAEs), would cause algebraic loops due to variables related only through an algebraic equation. It is argued that this method becomes increasingly more difficult and error prone for larger and more complex system models [61, 81]. Additionally, having undergone this manual transformation, there is a loss of correlation between the simulation model and the physical domain, since the individual components can no longer be uniquely identified.

With the advent of digital computers in the 1960s, many simulation programs were produced to see if analogue simulators could be replaced. Initially these programs made use of the same analogue simulation diagrams as textual inputs and mimicked the operation of the analogue devices. This was partly due to the fact that numerical integration techniques for solving ODEs were well established at the time, and also it allowed for the continued use of tried and tested practices from analogue simulation. In this way the analogue paradigm was transferred to the digital domain.

A standard known as the Continuous System Simulation Language (CSSL) helped in collecting the various efforts and defining some common concepts for simulation programs [80]. According to the CSSL standard, there were three ways in which a system could be represented; interconnected blocks, mathematical expressions and programming constructs. As technology advanced and graphics capable computers became more common place in the 1980s, several graphical block modelling tools became available. Simulink is one such tool that has gained widespread use in both educational and commercial fields. These tools introduced the ability to build models visually by putting together blocks, stored in libraries, with inputs and outputs which can be connected by drawing lines between them. However, the fact that the analogue paradigm is still in use with these programs and that there is a need to describe the system in state-space form, gives rise to some problems such as [58, 61, 80]:

- connections between blocks have a fixed causality (data transferred in one direction),
- the model is still not visually representative of the system being modelled, and
- the use of physics based model descriptions is difficult (ODEs must be used in favour of DAEs).

However, it must be recognized that this approach has been successful and given

Table 3.1: Domain Specific Modelling Tools.

| **Domain** | **Modelling Tools** |
|---|---|
| Electronic | SPICE, VHDL-AMS |
| Mechanical | ADAMS, SIMPACK |
| Process | SpeedUp, gPROMS |

rise to many industry standard tools in specific domains. Domain-specific tools are specialised and provide users in that domain with a library of components from which they can easily build their models. Some examples of these tools are listed in Table 3.1. These tools demonstrate the benefits of user friendly interfaces with visual modelling front-ends, the use of component libraries and optimised numerical methods. However, it is argued that the downside of these tools is their lack of flexibility and adaptability when a problem does not exactly fit the tool as in the case of mixed-domain modelling which is prevalent in modern technical systems such as a HEV [82, 83]. Additionally, modification and customisation of such domain-specific tools is often a challenging task. Some of the domain-specific tools and languages such as the general PROcess Modeling System (gPROMS) and an extension of VHDL enabling analogue and mixed-signal system design (VHDL-AMS), have provided increased levels of flexibility by moving away from the analogue paradigm in favour of non-causal modelling with object-oriented concepts [80, 83].

### 3.1.2 Object-Oriented Physical Modelling

Physical modelling refers to a modelling approach where the natural physical laws of components are used to define a system. System behaviour is then deduced from the application of these laws and their interaction with one another. In object-oriented physical modelling, a system can be modelled by connecting its constituent components together. Using this approach to modelling leads to a new paradigm where the model acts as a constraining relationship between system variables.

In the late 1970s, a language called the Dynamic Modeling Language (Dymola) made use of physical modelling principles to try and avoid the problems encountered by the existing analogue paradigm based tools. The idea was to reduce the gap between the system description and the model description. The Dymola language drew on features from the first object-oriented language Simula, and symbolic computational

methods. By defining model classes and connection types such as shafts and wires, along with the variable constraints for those connections, it was possible to build a model in any domain using the same methodology. Furthermore, the model equations were presented as DAEs and symbolic manipulation used to convert these DAEs into ODEs. The Dymola project was postponed at that time, since computing memory limitations meant that it could not be used to solve problems in the order of 300 or more equations [80].

Technological advances in the 1980s as well as progress in the fields of numerical integration for DAEs and object-oriented software caused a revival of interest in object-oriented modelling (OOM). In 1989 the object-oriented language Omola was developed which allowed for models to be broken down into a series of interconnected components in a hierarchical manner. Interaction between components was defined at connections and class structures were used for defining the components. Additionally, an interactive modelling environment OmSim was produced and a complete tool formed that provided a graphical interface, symbolic manipulation, numerical solvers for ODEs and DAEs, and plotting of results.

Progress with Dymola, Omola and other similar object-oriented and equation based modelling languages that underwent parallel development during the 1990s, such as ObjectMath, SIDOPS+, Allan and Smile, led to an international initiative for defining a uniform modelling language. The developers of these languages brought together their combined experience in order to unify and standardize the concepts and structures required in an object-oriented physical modelling language [58, 84]. As a result of this effort the generic modelling language Modelica was produced, with the first ratified version being released in December 1999.

Modelica is a freely available mathematical modelling language intended for modelling complex systems in multiple domains [84]. It uses an equation based and object-oriented approach to physical modelling and can support various formalism such as DAEs, ODEs, bond graphs and Petri nets. The main motivation behind its development was to enable models to be built in a standard way so that they could be exchanged and reused between different users and domains [80]. The first commercial tool to support Modelica usage was Dymola, this time meaning Dynamic Modeling Laboratory, which was soon followed by MathModelica. Modelling and simulation tools such as these provide the following generic features [58]:

- a Modelica translator,
- a code compiler,
- a run-time simulator,
- a graphical modelling environment with access to model libraries,
- a textual editor for scrutinizing the Modelica code.

Åström et al. [80] point out that although the technology advancements started occurring in the 1980s, it was not until the mid 1990s that the need for a paradigm shift was recognized. A new object-oriented modelling approach was then taken based on non-causal equation oriented modelling which allowed for increased levels of sharing and model reuse.

## 3.2 Object-Oriented Modelling in Industry

The use of OOM principles is by no means new to the field of engineering. This section presents some motivations and benefits of applying the OOM approach in different industry sectors, with the aim of highlighting key concepts and features to support HEV development.

### 3.2.1 Automotive Sector

Hong et al. [79] make use of OOM techniques to develop a powertrain model for a vehicle with a petrol engine and automatic transmission. The purpose of the model is the design and evaluation of the powertrain controller. Initially a hierarchical abstraction is performed on the powertrain in order to reduce the complexities of the design task, to give an intuitive structure to the model and to allow for a more modular programming approach. Further, with each module defined as an individual object, the property of reusability of objects is used to reduce programming effort. To implement this work, the authors make use of the Matlab and Simulink environment. The powertrain is initially abstracted into three independent classes; namely the engine, transmission and driveline. Each of these top-level classes are

developed independently of each other and abstracted further. For example the engine is comprised of modules for the fuel injector, intake manifold, throttle body and torque production. Simulation results are shown to agree with experimental results and the model is used in refining the integrated engine and transmission control.

In [79] the author's state that not only physical parts are treated as objects, but also equations and algorithms. This is most likely due to the choice of modelling environment, since all aspects must be represented in a block diagram form. Another point that is not dealt with by the authors is that of causality. Using block diagrams models such as those in the Simulink environment allows for information to flow in one direction from a predefined output to a predefined input. In order to cover all possible causalities between interconnected parts, a database of several options for each class would be required with a precise description of required neighbouring classes. This would then reduce the programming effort benefits by adding more complexity and modelling requirements. It is possible to eliminated this particular problem through the use of an OOM language where objects are non-causal.

### 3.2.2 Industrial Automation and Process Control

The application of OOM in the field of automation and process control is discussed in [85]. Understanding the overall system behaviour requires the modelling of the physical process objects in conjunction with control system objects. However, the typical models used in these two areas, such as component models and block diagrams, are not easily integrated to form an overall system model. Maffezzoni et al. [85] show that while object-oriented properties such as modularity, encapsulation of model equations, independence from external connections and aggregation of sub-models are easily and intuitively related to the process models, the applicability to control system modelling is not as obvious. The abstraction and aggregation of control systems is largely dependent on how the system is analysed. For example the system can be considered in terms of its functional behaviour, its architecture or its software structure. Typically functional models are used, meaning control systems require causal modelling unlike the non-causal process components. The authors suggest that OOM is well suited to modelling control systems since the modularity can be used to separate functional and behavioural features, and a hierarchical control structure can be defined with supervisory control at the top level and sensors and actuators in the lower levels. An OOM language can then be used to develop

both the non-causal process or plant models and the causal control system models which can be integrated in a complex model by connecting the two model types via sensors and actuators. Maffezzoni et al. also highlight the fact that the OOM approach leads to very large model descriptions for complex systems which would lead to unreasonably long calculations of the numerical solution during simulation. However, this problem can be avoided if the simulation environment is capable of performing symbolic manipulation on the model, thereby reducing the number of equations and overall model order before attempting to determine the numerical solution [85].

### 3.2.3 Aerospace

In the aerospace sector, modelling and simulation tools are critical in assuring safety through the correct functioning of the control systems. When considering spacecraft or multiple satellite missions, the complexity of the control task and the high cost of failure, demand efficient modelling tools in all of the applicable domains and for all stages in the development cycle. Pulecchi et al. [86] claim that though there are several commercially available tools that provide the required functionality in some areas, there is no unified environment that is sufficiently flexible to support the entire development cycle. In this respect, the authors propose a unified modelling approach based on object-oriented concepts. To implement this work, Pulecchi et al. favour the use of the non-causal, object-oriented modelling language Modelica, due to its systematic approach to modelling and the ability to support multi-domain problems.

Modelica is used in [86] to create a library of classes describing the spacecraft dynamics, sensors, actuators and controllers for developing a spacecraft model. Flexibility is achieved through the creation of reusable classes and the modularity provided by the use of objects. Further, classes can be updated in terms of modelling accuracy and complexity as the development progresses. The developed model library provides sufficient flexibility for performing initial design and sizing studies as well as detailed control system simulations. Pulecchi et al. illustrated this by presenting results of three separate case studies. In the first study, a higher level of abstraction is used during the preliminary design stages to model the spacecraft and asses the external disturbances on it. The non-causal nature of the modelling language is important here as it allows the modeller to provide the desired orbit and trajectory

and let the simulator determine control forces required to remain on that trajectory. For the second study, a higher fidelity model is required that takes structural dynamics into account in the form of flexible appendages. To do this the rigid body of the spacecraft model is modified by connecting it to another rigid body via a flexible beam component. Additionally, this spacecraft model is used to compare the performance of attitude control using different types of actuator models. Finally, the third study considers the validation of different control design approaches for the attitude controller in the case of a "swarm" of three identical satellites. In this case the satellite model is built through specialization of the spacecraft base class. Particularly the actuator model is replaced by three orthogonal coil and thruster assemblies, the sensor model is replaced with a multi sensor configuration and the control model is replaced by the swarm attitude control system being tested.

Yu et al. [87] develop a simulation environment at the McDonnell Douglas Corporation for the purpose of modelling aircraft logistics and support systems. Working from the premise that the available simulation and modelling tools made modelling a difficult and time consuming activity that often led to inefficient, unrealistic and costly models [87]. An OOM based tool is proposed in order to address the shortcomings of the then popular mathematical simulation languages such as Simscript and SLAM. More precisely problems such as the absence of formalisms for including domain specific knowledge, inefficient handling of increasing size and complexity of system models and the rigidity of the modelling language. The main objective of the authors is to provide a generic environment that offers a close link between real and model objects, allows for hierarchical modelling and offers a simplified method for modifying object behaviour. Yu et al. [87] claim that the implemented OOM approach allows the users to concentrate their effort on understanding the system as opposed to focusing on the simulation language. Further, the object based graphical developed environment provides increased modelling flexibility and reduces the development time.

### 3.2.4 Building HVAC

Modelling and simulation is intrinsic to the understanding of building heating, ventilation and air conditioning (HVAC) and developing the associated control systems for maintaining the indoor bioclimate. Comparative studies of OOM and procedural modelling approaches for the analysis of thermal flows, radiation and ambient tem-

perature are performed in [62, 81, 88]. These are discussed further in Section 3.3.3.

### 3.2.5 Summing Up

This section has shown that OOM techniques have been successfully adopted to deal with the complexities arising in a variety of industrial applications. Further, the techniques have been applied to deal with both logical complexity and model flexibility in industries facing similar modelling challenges as in HEV development. This implies that these techniques can be applied to task of evolving the HEV design space in a more efficient and manageable manner. Also, the fact that traditional modelling tools in the automotive sector are being used to explore the benefits of OOM, suggests that the use of object-oriented modelling tools can be integrated into the existing development methodology.

## 3.3 Key Features of the Modelica Modelling Language

For a modelling language to represent a real-world system by means of mathematical models, it requires the following two features [76]:

1. *Structuring* abilities for modelling complex component combinations as discussed in Section 3.3.1.
2. The ability to incorporate the necessary equations to describe the *system behaviour* as discussed in Section 3.3.2.

Modelica meets these requirements since it is both a declarative and object-oriented modelling language. For example, in Modelica, the *equation*

$$F = ma \tag{3.1}$$

describes the fundamental relationship between force, mass and acceleration, and thus also has meaning if $F$ and $m$ are known and $a$ is unknown. Hence, in Modelica, equations are stated in a neutral form meaning that it is not necessary to consider the computational order [60].

Structural modelling languages are based on imperative programming. For example, in Matlab equation (3.1) is seen as an *assignment* which describes how the force variable ($F$) can be calculated for a known value of $a$.

By not enforcing how a models behaviour should be calculated and only describing what it should calculate, the declarative language provides more flexibility [76]. Additionally, this can save time in model development since the designer is not required to manually transform equations for the system being modelled into state-variable form and can rather concentrate on the logic of the problem [60, 89].

The following subsections aim to identify particular features of OOM languages that aid in the design of modular, flexible and reusable models with emphasis on those features appropriate to HEV modelling.

### 3.3.1 Model Structuring Features

This section describes the Modelica features that are most relevant for presenting the structure of complex models. Further exploration of these attributes is provided through examples in Chapters 5 and 6, Sections 5.2 and 6.2 respectively.

1. **Models** – These are the basic building blocks within the Modelica language and are equivalent to classes as described in appendix A.1. A model is used to describe the properties and behaviour of components and can contain instances of other component models.

2. **Hierarchical modelling** – A complex system model can be built by aggregating simpler components. This can be done on several hierarchical levels giving the model a structured design. For instance, on the top level, a HEV model could contain an electric motor component, engine component and a chassis component. In turn, looking a level deeper, the motor model could contain resistor and inductor components. Each component made up of other components is known as a structured component.

   An important consideration is the connection between each component on each level. In Modelica connectors are used to allow components to interact with each other. A connector is itself an instance of a connector class, which is a type of class that only contains variables and no equations. Connectors represent

the actual "real world" interaction between components such as electrical or rotational couplings. Each type of connector is defined by a variable that is equal at the connection, known as the across or effort variable; and a variable that sum to zero at the connection (through/flow variables) [2, 90]. Listing 3.1 shows an example of an electrical connector with `Voltage v` defined as the across variable, and `Current i` defined as the `flow` variable.

```
connector Pin "Pin of an electrical component"
  SI.Voltage v "Potential at the pin";
  flow SI.Current i "Current flowing into the pin";
end Pin;
```

Code Listing 3.1: Connector class for electrical components.

3. **Inheritance** – The object-oriented property of inheritance is the main contributor towards class reusability. In Modelica a child class such as a component model can inherit from a parent class or base model by using the keyword `extends`. Often, at the lowest hierarchical level, abstract classes are used as the parent class. The Modelica equivalent for the abstract class used in creating base models is the partial model, denoted by the keyword `partial`. A major benefit of inheritance is that it allows the developer to produce one piece of common code which is shared by multiple models, thereby making maintenance easier [76]. In HEV development studies it may be necessary to test different electrical architectures while maintaining the same mechanical architecture. In this case the mechanical architecture can be defined in a base model that will be inherited for each of the different electrical architectures.

4. **Types** – The ability to define data types is very important as it helps to reduce ambiguity in model construction. Modelica provides the user with the ability to define any needed variable types as in the following definition of a `Mass` type:

   ```
   type Mass = Real(quantity="Mass",final unit="kg");
   ```

   The type definition also allows for quantity and unit variable definitions which can be used by the compiler to perform dimension checking on the model. Also effort and flow variables used in connector definitions are instances of a specific type. For example, in Listing 3.1, the code line `Voltage v` means variable `v` is

an instance of type `Voltage` and will thus have units in volts. This is important in the development of HEVs considering the number of types occurring across all the engineering domains combined in a HEV powertrain design.

5. **Class Parameters** – Instances of a class within another class can be declared replaceable using class parameters. In Modelica class parameters have two forms, *instances* and *types*. With instance parameters, components are declared as `replaceable` inside the class definition. When that class is used, the replaceable components can then be exchanged with compatible components by using the keyword `redeclare`. With type parameters the type of a component is declared replaceable before instantiating the components in the class. In this way all components of that type can be replaced by another type at once by redeclaring the type parameter. For instance, in order to test the use of different electrical machines in an HEV design, the electric machine model can be declared as replaceable, allowing for example a synchronous motor to be replaced by an induction motor before simulation.

### Design of Model Libraries

Model library design refers to the structure of the models, sub-models, components and partial models; and also how these models are hierarchically linked. The structure of the library is chosen in order to simplify the understanding and usability of the library for different potential users and is thus very dependent on the complexity of the system being modelled and the scope of the modelling exercise. Library structuring is a key part of the modelling method proposed in this Thesis as described in Section 4.4.2 and its importance as a model management task is stressed again in Section 7.1.1.

In the case of a HEV powertrain model, the system is complex with various domains being represented and possibly various parties developing component models. Since different domain specialists may already have their own libraries for their respective domains, a new library for HEV powertrains could be defined to best make use of these libraries or to provide a generic approach for investigating different architectures.

Containers for similar classes, from models to partial models, are referred to as packages and sub-packages. Packages help to prevent name conflicts in code by

keeping related functions and classes together, and adding the package name as a prefix to all definitions within the package. For example the line of Modelica code:

```
HEV_Modelling.BaseSubsystems.PowerSupplies.FuelCellBase fuelCellOne
```

creates an instance of the `FuelCellBase` model called `fuelCellOne` and the name prefixes indicate that his model is contained within the `PowerSupplies` subpackage of the `BaseSubsystems` package in the `HEV_Modelling` library. Or rather, this line of code is created when an object from that subpackage of the library is added to the model. This, together with formal naming conventions, is important for promoting compatibility when large scale models are being developed by different parties [58].

Hierarchical composition is also very important since if it is not well planned, it can lead to rapid growth in library size which is cumbersome to work with and time consuming to manage. Tummescheit [76] points out that the overuse of inheritance when structuring libraries is one of the main causes of oversized libraries that are difficult to understand and use. This is because it leads to the creation of many unnecessary intermediate partial models. For complex systems where many models and sub-models are required, a better approach is to use a mix of inheritance and aggregation in order to describe the system composition. In this way inheritance can be used when there is a general commonality between several components or models such as all car models inheriting a chassis model or electrical components inheriting electrical connectors. Aggregation is then used to add extra components to existing models in order to provide for variety and fidelity changes. Together with the use of the replaceable class parameter, this strategy allows for instance, several tyre models to be added to the chassis model which can be replaced by either linear or non-linear model variants.

### 3.3.2 Model Behaviour

This section discusses the Modelica features that are most pertinent to describing the model behaviour. Further exploration of these attributes is provided through examples in Chapters 5 and 6, Sections 5.2 and 6.2 respectively.

1. **Equations** – A non-causal declarative form for expressing the underlying physical laws that describe a components behaviour. These can be in the

form of ODEs, DAEs or difference equations. It is important to point out that while in object-oriented programming it is possible to override the methods used in classes, in Modelica the methods are replaced by equations which can't be overridden. In Modelica it is however possible to achieve this effect by making components or models replaceable and then redeclaring them before use. This feature is very useful, for instance, in defining different levels of fidelity for the same object.

2. **Algorithms** – Usually equations are most useful for modelling physical systems but in some cases there is a need for the use of imperative assignment statements and logic statements such as $if - then - else$, $for$ and $while$. Algorithms are executed in the order they are stated. In particular, algorithms are useful when implementing discrete logic control in the same manner as in the control hardware. Algorithms may be used to model the logic in a supervisory controller of a HEV, where different control strategy can be implemented depending on different selected vehicle operating modes. For instance if a driver selected sport mode in the vehicle the strategy for battery usage may change from charge-sustaining to charge-depleting in order to provide increased performance.

3. **Functions** – Algorithms can be encapsulated in functions. Many mathematical functions, such as $sqrt(), mod(), sin()$, are provided by Modelica's standard *math* library. It is also possible for a user to define functions with specific inputs and outputs calculated from encapsulated algorithms. The use of functions together with equations does not constrain the causality as inputs can still be calculated from a given output.

4. **External functions** – This feature of Modelica provides the user with a means to include functions from external software libraries using either C or Fortran. Thereby providing flexibility to the user through the use of existing legacy code. It is also important in promoting code reuse and knowledge sharing where possibly much time and effort has gone into developing complex model descriptions in an external or domain specific environment. Together with the need for sharing code in collaborative development efforts, is the need for protecting the intellectual property contained within certain models. These aspects of code sharing and protection are discussed further in Section 7.1.3.

Through a combination of these features it is possible to fully describe any physical system in terms of its mathematical behaviour. From an object-oriented perspective, each object's behaviour is determined by the equations, functions and algorithms used to define the operations in that object's class. In Modelica the component behaviour is defined by the equations in the model.

### 3.3.3 Compiling and Simulating Models

It can be said that the natural world and therefore physics is non-causal [61, 81, 91] and that causality is invented in order to simplify computation. One of the main features distinguishing new equation-based modelling tools from the more traditional block-oriented tools is the way that they deal with causality. In the case of block-oriented modelling, it is the task of the modeller to decide upon the desired causality before hand and then proceed to develop the models with this causality in mind. Considerable effort is required at this stage to describe the physical laws governing each system and component in a manner that fits the chosen causality in order to make computation possible [81]. With equation-oriented approaches the modeller must decide what objects are needed to describe a system and how they interact. The effort is then placed into describing the physical laws of each object and its interactions, which can be done using DAEs without worrying about the computation order. It is then the task of the translator to transform the model equations into a computable order based on the given model parameters.

Wetter and Haugstetter [62] claim that the model development time in procedural modelling is between 5 and 10 times longer than that associated with equation-based methods. While Zupančič and Sodja [81] argue that not only is the traditional method more time consuming but it is also more error prone and open to the creation of algebraic loops which need to be dealt with by the modeller. However, this view is slightly skewed by the use of library components with different levels of model fidelity already being available. That is to say that the basic blocks available to start modelling in the equation-based tool may be at a higher level of abstraction than those in the block-oriented tool. It can be said though, that the equation-based method is more conducive to the creation of reusable library components. Also, since the causality is not of interest during development time, library components can be developed independently from a specific modelling project. The ability to develop a single subset of models is particularly advantageous for HEV modelling

due to the great variety of technology options and architectures.

In terms of computation time Wetter and Haugstetter [62] state that the equation based approaches are 3 to 4 times slower than procedural approaches. Sahlin et al. [88] point out that although the superior numerical performance is often quoted in the building industry as a reason for staying with procedural modelling tools, the comparison is not straight forward and often misinterpreted. Consideration should be given to the level of modelling and the accuracy of the solution at each timestep [88]. Further, it must be remembered that the procedural tools usually have specific numerical solutions for a particular problem domain while the equation-oriented tools use general DAE solution methods. Also, it is noteworthy that numerical techniques for solving ODEs were well established by the 1960s while methods for solving DAEs only started appearing in the 1970s and are thus still improving [80]. A brief description of numerical techniques is provided below for completeness.

### Numerical Techniques

Traditional modelling leads to a system description in ODE form that have the explicit state space form shown in equation (3.2).

$$\dot{x}(t) = f(x(t), u(t)) \tag{3.2}$$

where $u(t)$ represents the input variables to the system and $x(t)$ represents the state variables with time derivative $\dot{x}(t)$.

ODEs can usually be solved very efficiently by explicit integration methods such as the Euler and Runge-Kutta method. A method is said to be explicit if the next state is calculated from one or more previous states [58]. However, explicit methods tend to have small regions of stability and can become unstable if the integration step size is outside of this region and therefore produce incorrect solutions [58]. This is of particular concern when the system description contains stiff differential equations which require implicit forms of the integration methods to be used. According to Moler [92] computational stiffness can be defined as follows:

> "A problem is stiff if the solution being sought varies slowly, but there are nearby solutions that vary rapidly, so the numerical method must take small steps to obtain satisfactory results."

Implicit methods solve an equation involving both the current and previous step in order to compute the next step. Further these methods have larger stability regions than the explicit counterparts and are less computationally intensive when dealing with very small time steps. The stability region can be defined as the time step region within which a particular numerical method's error will remain bounded [93]. In other words if the time step is outside of this region, that method will not succeed in finding a precise solution. Moler [92] also argues that stiffness is mainly a time efficiency issue since it is possible to find a solution with a non-optimized method, only it will take a considerable amount of time. Sometimes it is necessary to use variable step methods such as the Adams method in order to more efficiently find a solution for stiff systems. These methods allow for the integration step size to be changed during simulation.

For object-oriented modelling, since each object is defined independent of its context in a non-causal or declarative manner [94], the system is described as a series of connected differential and algebraic equations. DAE systems have the general implicit form shown in equation (3.3).

$$f(x(t), \dot{x}(t), u(t), y(t)) = 0 \tag{3.3}$$

where $y(t)$ represents the output variables.

A compiler then extracts all the object equations within a model and the equations for all the connections to produce a flattened list of equations, constants and variables. This list is then sorted and manipulated to give a computational causality based on data-flow dependencies which gives a system of high-index DAEs. In general OOM and in particular mechanical and mechatronic modelling lead to DAEs of index 3 [58]. Currently there are few numerical methods for solving DAEs with index higher than 2. The most common method for solving index 1 DAEs is the differential algebraic system solver (DASSL) which is an implicit variable step method as are most DAE solvers. Therefore it is essential that a multi-domain modelling and simulation tool also has the ability to perform a mixture of symbolic (see section 2.2.4) and numerical techniques in order to reduce the DAE index to 1.

It is possible to reduce the index 1 DAE even further to an ODE (index = 0) but not without a considerable increase in computation. Cellier [61] argues that there is minimal benefit in converting DAEs into explicit ODEs when dealing with large complex models since these systems are invariably stiff and therefore require implicit solvers.

## 3.4 Concluding Remarks

Modelling and simulation in the area of HEV development has similar problems to those in other industries such as:

- complexity of design,
- the system is described by multiple engineering domains,
- control is an integral part of the system,
- the need for reduction of development time, and
- the need to reduce cost of errors through early detection.

OOM is a reasonable approach to help cope with or overcome these types of issues, as shown within other industries. However, in order to provide the most flexibility across all domains and to support as much of the development cycle as possible, the modelling environment should be implemented with a truly object-oriented language. This is in contrast to applying a OOM methodology in a procedural environment through a component based interface. It is necessary to be able to create models from non-causal objects to maximize efficiency and to allow for parallel development in different domains by the respective specialists.

The use of an equation oriented language is preferable in reducing the complexities of the modelling task as this allows for the direct use of physical equations in modelling without the need for user manipulation and aids in the creation of non-causal models. However, as discussed earlier, the use of such a modelling language makes it necessary for the simulator to be able to implement specific symbolic and numerical techniques. Finally the use of an object-oriented language for the entire modelling and simulation tool makes it easier for data to be shared between different tools within the development chain and therefore aids in reducing errors through computer aided engineering (CAE), as is discussed in Section 7.1.1.

# Chapter 4

# Object-Oriented Analysis and Design for HEV Powertrain Modelling

Referring back to the Venn diagram presented in Figure 1.1, this chapter deals with the overlap of the HEV powertrain modelling and OOM elements discussed in the Chapters 2 and 3 respectively. Further, it also introduces the third element of systems engineering to the Thesis.

## 4.1 Proposed Solution

Recalling the problem statement in Section 1.2, a possible solution is the use of OOM principles in order to perform a more robust evolution of the HEV powertrain design space. An object-oriented design environment can provide benefits to both developers and end users. In recent years, OOM techniques have been viewed as a possible solution to model management issues for alternative vehicle development such as HEVs. OOM promises higher flexibility in testing concepts, component design and control strategy development [2].

Management of the many design models requires a well structured modelling approach. This approach must allow different users within the design process, such as those who specialize in component design, vehicle analysis and controller design,

to share and reuse models and data. Systems engineering design and management principles can also be used to help manage the complexity of the HEV powertrain design process.

This Thesis proposes a model-based design approach making use of systems engineering principles and object-oriented software development methods for HEV powertrain modelling, in order to deal with the increased model complexity and variety within the growing HEV design space. In particular a modelling method is presented that can be used to create flexible libraries for HEV powertrain design. This method is intended to be used in conjunction with a systems engineering development framework in order to increase the reliability, efficiency and speed of new HEV developments.

An alternative to this solution, making use of traditional causal block oriented models, would require an appropriate interface for grouping multiple component models. This interface would have to keep track of both the various causalities of each model as well as the different levels of fidelity for each causal implementation. Additionally a database or code structure would be required to keep track of the types of inputs and outputs that each component used in order to prevent "impossible connections" occurring. Further, a strict methodology would be necessary for building and maintaining models so as to ensure model compatibility and propagation of model changes to all relevant models.

## 4.2 Software Development and Modelling

It is important to understand that OOM and design does not refer to a specific domain or tool but describes a means of approaching problems by relating models to real-world concepts [95]. The applicability of object-oriented techniques to the field of simulation and modelling stems from the fundamental OOM principle of thinking in terms of physical objects so as to promote a clear link between program design and the real world [96]. This chapter presents the reader with some background on traditional object-oriented design techniques used for designing and developing complex software systems. Ideas from these approaches are then used in order to formulate a development method for the modelling of HEV powertrain systems and exploring the associated complexities. It must be pointed out that the development

methodology used in software and systems engineering refers to a set of procedures or methods to be implemented throughout the life-cycle. Therefore, the model development method proposed in this research is meant to be used as one of the procedures within the framework of any given system development methodology.

### 4.2.1 Software and System Modelling Languages

Due to the growing interest and popularity of the object-oriented paradigm in the late 1980s and early 1990s, different researchers and experts published methodologies promoting their own approaches [97]. Furthermore, each used different terminology and notation for describing the concepts of design while focusing on a preferred language or application domain [8]. By the mid 1990s there were over 50 methodologies for object-oriented analysis and design such as Booch [78], Rumbaugh [95] and Jacobson [98] methods, a period described in the literature as the "method wars" [99]. Describing the differences between the numerous development methods is beyond the scope of this Thesis, the interested reader is referred to Fichman and Kemerer [100] and Graham [97] for more detail. Fichman and Kemerer [100] compare the traditional structured methodologies with several of the object-oriented methodologies and conclude that though object-orientation was still evolving and no methodology had become standard, it was based on "powerful ideas...that have firm theoretical foundations" and was considered the better approach by leading programmers and academics.

Along with the abundance of methodologies was an equally large amount of different modelling languages, or diagrams and notations for use with respective methods [97, 101]. At the time Booch, Rumbaugh and Jacobson, the developers of three prominent methodologies, were all working for the same software company and were tasked to unify their various terminologies and notations in an effort to produce a general purpose object-oriented modelling language [102]. Ultimately this led to the development of the Unified Modeling Language (UML), which was subsequently adopted as a standard by the Object Management Group (OMG) [95, 103]. The UML standardized a means for specifying, visualizing, constructing and communicating software system structures, and aided in the further spread of object-oriented techniques [102, 103]. It is worth noting that the UML was designed to be compatible with the leading object-oriented methodologies and after its release gave rise to a customizable software development framework called the *Unified Software Develop-*

*ment Process* which was based on best practices of leading software companies.

The ability to visually represent the structure of a system throughout its development life-cycle is not only useful in the software domain but in other engineering domains too. Object-oriented descriptions are quite easily applied to physical systems since the physical system can be described in terms of its parts and their constituent components. For example, in [79] the author uses a UML class diagram to represent the abstracted architecture of a vehicle powertrain into engine, transmission and driveline subsystems; and in turn the subsystems are abstracted into components. Rachuri et al. [104] make use of the UML to create a model for representing electro-mechanical assemblies with the scope of facilitating the exchange of knowledge and information between different systems. The authors of [104] propose to achieve this by using the UML model as an integrated information model that all stakeholders can use and contribute to.

In the automotive industry, where typical projects involve the collaboration of multiple parties encompassing engineering skills from a variety of domains, there is a need for the availability of high level system models. This is particularly useful for aiding the design of distributed control systems where developers could make use of such a system model in order to understand how their control functions will integrate with the overall system [105]. In this respect, the UML is a useful tool as it offers a set of semantics designed for sharing information between different stakeholders. Though it can be argued that the UML cannot adequately represent all the requirements of a complex engineering design, it does allow for customisation and extension of its semantics.

### UML Extensions

The UML provides a means of customising its usage to meet particular needs by extending its definitions to either modify existing or add new constructs, creating a so called UML profile. In order to support model-based design, the OMG developed a UML profile called the Systems Modeling Language (SysML) [106]. The SysML is a general purpose language with constructs for modelling systems engineering problems. It is based on a subset of the UML and extends its functionality to support the systems engineering activities in the development life-cycle [107]. Further, the SysML supports the specification, analysis, design, verification and validation

of multidisciplinary complex systems through the use of nine diagram types [108]. These diagrams allow for the representation of the structure, behaviour, requirements and parametric relationships of systems [108].

The use of the SysML is not without its shortcomings and may not be the correct profile for meeting all the modelling requirements of complex mechatronic systems. Marco and Vaughan [105] highlight problems with navigating between the many diagrams created in the SysML and the transparency of the links between these diagrams, also the use of software-centric semantics does not aid in communication when many of the parties involved are not from software related domains. Johnson et al. [106] point out that while the SysML is good for modelling large and complex engineering systems in multiple domains, it does not provide a means for explicitly modelling continuous system dynamics in a non-causal manner. Instead of developing a new UML profile, the authors propose a bidirectional mapping between the SysML and the Modelica language in order to be able to benefit from the features of both languages. In particular, the information models from the SysML can be augmented with behavioural models capable of describing physical system behaviour in a non-causal way using Modelica semantics.

In contrast, Schamai et al. [109] set out to achieve a similar goal by developing a new UML/SysML profile called the ModelicaML. This new UML profile reuses features from the UML and the SysML, while adding additional language constructs for including Modelica concepts such as equation and simulation diagrams. Like the SysML, the ModelicaML is only a graphical notation and not directly executable, however, it is possible to generate Modelica code from the ModelicaML models [109].

With regards to this Thesis, no system modelling has been performed within the UML or any of its derivatives. This is because the primary focus of this work is on the physical modelling stage of the design process. However, the importance and benefit of a systems level model is well noted by the author, both with regard to the management of a modelling project as discussed in Chapter 7, and further work on this area as mentioned in Chapter 8 Section 8.1. Generic UML class diagrams are used for illustrative purposes to help the reader visualise the links between the software and physical modelling domains. However, it is recognized that the availability of an overall system model forms a necessary medium for communicating and managing information in a multidisciplinary project. As pointed out by Marco and Vaughan [105], the SysML provides an advantage when used to support other

modelling environments, by providing developers and engineers a means of exploring the system architecture through different levels of abstraction.

### 4.2.2 Software and System Development Approaches

Following the introduction of the UML and its popularity in the software development industry, several new software development methodologies were conceived such as the Dynamic Systems Development Method, Rapid Application Development methodology and Rational Unified Process. Each of these methodologies provides a framework within which the development activity can take place. Further, there are also many models for describing the development approaches that can be implemented within this framework. This section describes the three most commonly cited models in software and systems engineering literature, as these form the foundation for defining the requirements for more modern methods.

#### Sequential Development Model

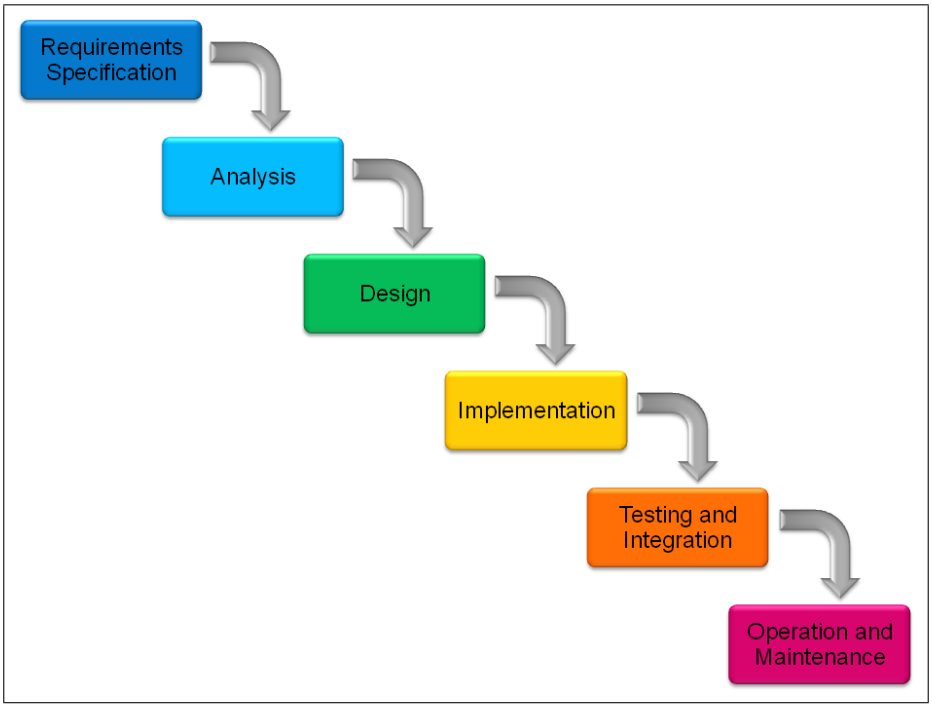



Figure 4.1: Waterfall development model stages [110].

One of the most commonly used development methods, especially before the unification of methods for the production of the UML, is the waterfall approach originally

described by Royce [110]. The stages of the waterfall model are shown in Figure 4.1. This approach follows a incremental sequence of events from the start of the development to the end, requiring each stage to be fully completed before moving to the next [8, 95]. The nature of this model made it easy to clearly define milestones between separate development stages, manage them and measure the time spent in each stage. This led to it being used as the basis of the US military standard for mission critical defence system software development [111].

Though intuitive, this method is not well suited to the design of large scale and complex systems, such as HEVs, where the requirements can change several times during the development life-cycle. For example, in an article by Wong [112] on the successful development of air defence system software, the author points out the inadequacy of the sequential model enforced by the Department of Defense standards. In particular, Wong describes software development as "*a complex, continuous, iterative, and repetitive process*" requiring iterative and parallel activities not reflected in sequential models. Further, the author proposes that a model based on overlapping incremental development allows for parallel and iterative coding, and more accurately depicts the complexities of software development.

It is noteworthy that though Royce [110] proposed that iteration should occur only between adjacent stages, even then, he recognized that this was not what occurred in practice. Often the first time a problem with the system design could be detected would be during the *testing and integration* stage near the end of the development process. This would then require a change in the *design* and possibly *requirements specification* stages too. Consequently, risk management and requirements gathering are two main reasons for the failure of this model [102]. More precisely there is a high risk involved with discovering major system design errors towards the end of the development and gathering complete requirements at the start of the project is unrealistic since requirements change and evolve during the system's development life-cycle. Having said this, the work done by Royce [110] in identifying the main stages in the development process is instrumental and is often cited further in more modern development approaches.

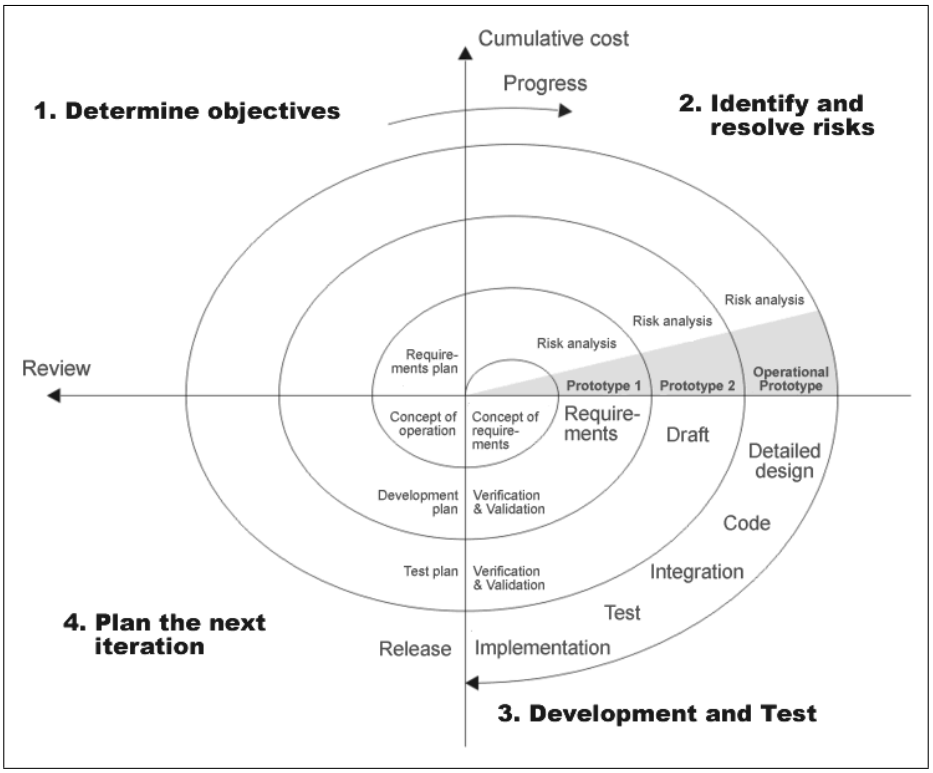



Figure 4.2: Spiral development model proposed by Boehm [113].

### Iterative and Incremental Development Models

Object orientation is used in order to simplify the handling of complex system design, and to promote the use and construction of reusable classes. Development of object-oriented software therefore requires a more iterative approach. From an automotive modelling perspective, a parallel can be drawn with the incremental development of higher fidelity physical system models. The sequential stages of the waterfall model are important and valid design stages, but need to be repeated incrementally as the various subsystems and components are developed. One approach developed to address the failures of the waterfall model is the spiral model proposed by Boehm [113].

Unlike the linear approach of the waterfall model, this approach considers development as a series of iterative cycles as illustrated in Figure 4.2. Development begins at the centre of the spiral in the top left quadrant and spirals outward in a clockwise direction with each complete cycle through all four quadrants representing an iteration. The four quadrants represent the main process actions in each iteration starting with objectives and constraints, then risk analysis, followed by development

and testing, finally ending with a review and plan of action for the next iteration.

One of the main benefits of this approach is its suitability for prototyping, where each complete cycle can be used to develop prototypes which are further developed with each cycle. Although it is possible to develop prototypes using conventional methods, together with object-oriented properties such as modularity and reusability, this can be achieved with less development time and effort [97]. Furthermore, prototypes can be used to gather valuable user feedback for the early realisation of requirements or specification changes. In a review of almost 500 software projects, over 40% of maintenance costs are attributed to changes in requirements which are either due to incorrect specifications or the need for adaptability in modern systems [97].

There are many accounts of the success and benefits gained when using object-oriented approaches over traditional procedural approaches. For example, the replacement of a procedural customer management system for Brooklyn Union Gas with an object-oriented equivalent achieved lower maintenance costs and a 40% code reduction (from 1.5 million lines) due to code reuse; or the upgrading of a procedural maintenance management system for General Motors where the object-oriented replacement required one twelfth of the development time and code size with an estimated 14:1 productivity gain [97]. In a comparison between prototyping and traditional development approaches, Alavi [114] shows that the use of a prototyping approach increased communication between the system user and the developer which in turn led to improved requirements specifications and increased satisfaction as to the final system performance. However, the management and control of the prototyping process is often seen to be more difficult due to the increased interaction of stakeholders and more frequently changing requirements [114].

#### Systems Engineering V-Model

Initially, as applied to software engineering, the V-model was used to avoid the problems associated with managing complex system designs by promoting early analysis and design [106], in the same way as the waterfall model. Forsberg and Mooz [115] enhanced this model to include the best features of both the waterfall and spiral models by adding development strategies which allow for iterative and incremental development. Each iteration of the process allows for the development

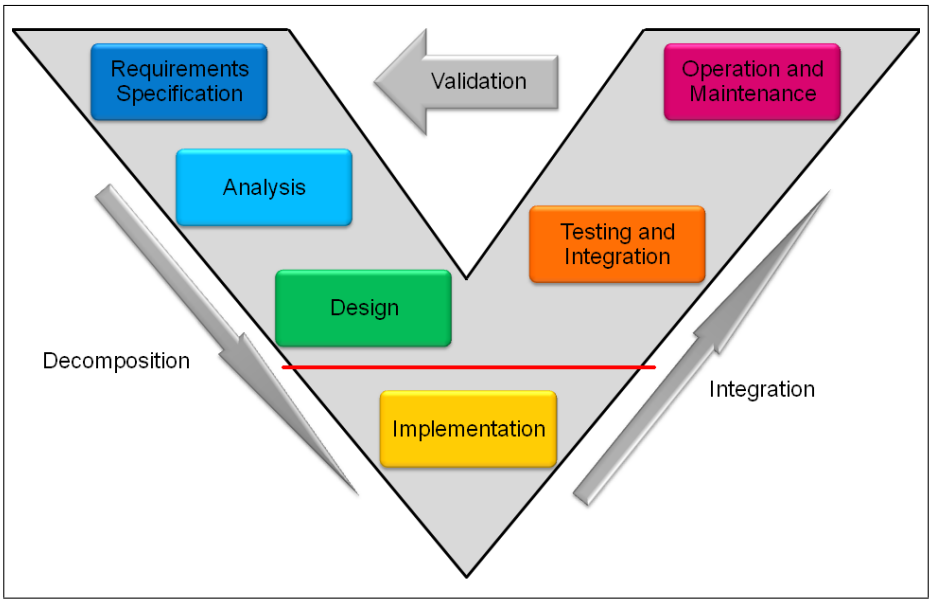



Figure 4.3: Iterative systems development V-model.

of new subsystems, thereby allowing the overall system to grow. It is also possible for certain independent subsystems to be developed in parallel if there is no clash with required resources. Additionally, the iterative process is more flexible to changes, provides more feedback on overall development progress with each iteration and therefore also minimizes the risk [95].

The iterative development V-model often used in systems engineering is shown in Figure 4.3. As the process progresses down the left side of the model the system definition and specification is found through decomposition. The red line in Figure marks the transition from the system engineering activity to the domain specific engineering activity where implementation takes place in the form of coding or building the defined subsystem. On the right side of the model the built subsystems are integrated into the overall system with validation results feeding back to the left side for the next iteration.

Colombi and Cobb [116] provide an example of how a systems engineering approach can be used together with rapid prototyping in order to meet critical user needs. Specifically, the attack controller lacked a means of quickly pinpointing the location of ground forces needed for directing an air attack. Colombi and Cobb [116] combine the use of the both V-model and spiral model in order to provide incremental prototype development as well as parallel development of alternative solutions in order to minimize project failure.

Irrespective of the chosen development model, it must be remembered that they are models and are therefore approximations of the real development process. These models serve as guidelines from which lessons can be learned, there is no model that can precisely describe an error free development for all projects without the need for adaptation to the particular problem being considered. However, the stages followed in the development life-cycle are common to most models with main differences being in the iterations between the stages or time spent at a particular stage. These stages are listed below. A short description for each of these stages can be found in Appendix A.2.

1. **Specification of requirements**.
2. **System analysis**.
3. **Design**.
4. **Implementation**.
5. **Testing**.
6. **Maintenance**.

As illustrated in Section 3.2, the use of object-oriented techniques is not new to industry and the same is true for the use of system engineering methods. For example, Harrison et al. [117] describe an effort by the Ford Motor Company to implement system engineering techniques in order to improve the automation of powertrain assembly. Fisher [118] shows how models can be a better means of sharing information between systems engineers and developers through an example development of an on-board driver assistance system that performs various diagnostic and monitoring functions. Broy [119], on the other hand, describes how the rapid and complex development of embedded systems within modern vehicles not only requires software and systems engineering but that it should be tailored to meet the specific needs of the automotive domain.

In particular, with the increased dependence on digital technology and control software in modern vehicles for implementing features such as anti-lock braking and traction control systems, the automotive industry has for some time been making use of systems engineering development techniques. It is therefore argued by the author, that since systems engineering practices are already in use within the industry, making the transition required for complex HEV development using object-oriented methods in a systems engineering environment should not be that difficult a process.

## 4.3 Systems Engineering Standards

Seeing as the objective of this research is to provide a model development method that can be used within a systems engineering environment, it is appropriate to discuss some of the systems engineering standards at this point. This section is not intended to review the many standards related to software and systems engineering [120, 121], but rather to make the reader aware of the processes required for engineering a system.

Focus is placed on two particular standards, in order to illustrate how the proposed modelling method can be employed within the context of a systems engineering framework. The first, a standard developed by the Electronics Industry Alliance (EIA) called EIA-632 [122], discusses the fundamental processes for engineering a system. The second, developed by the International Organization for Standards (ISO) and the International Electrotechnical Commission (IEC) named ISO/IEC 15288 [123], defines a process framework for describing the life cycle of a system.

### 4.3.1 EIA-632 Standard

This standard groups the fundamental processes for defining and implementing a system into five categories. These are acquisition and supply, technical management, system design, product realization, and technical evaluation. Within each of these groups there are processes, as shown in Figure 4.4, from which an organization or department can select those appropriate in order to best fulfil their part of the system development process. These standards further specify a list of various requirements for each of the identified processes, which again are to be selected as required by the system developer.

EIA-632 defines a system as “An aggregation of end products and enabling products to achieve a given purpose” [122]. Where “end products”, are the product delivered to the customer or any user, and “enabling products” are those that enable the production and maintenance of the end products. From a hierarchical point of view, each enabling or end product, may have their own subsystems which need to be described in the same manner.

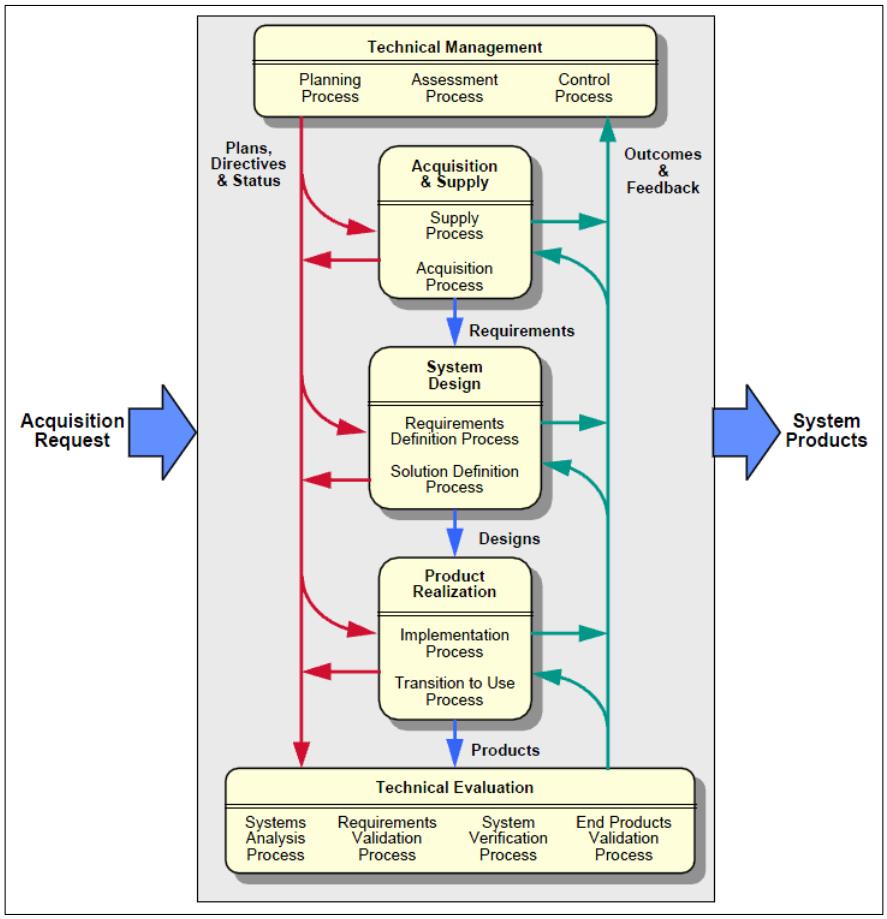



Figure 4.4: Categories and processes of EIA-632 [122].

Requirements are used as a means of decoupling the various development layers. The system design group of processes, at each layer of development, produce a set of specified requirements which become the assigned requirements for the next lower layer. Additionally, working back up the hierarchy, the lower layers feed back results in the form of data and enabling products, to the design and validation processes of the layer above. The relationship between the various requirements described in EIA-632 is illustrated in Figure 4.5.

Acquirer requirements refer to those requirements coming from either the customer or the user of a particular end product. These, together with stakeholder requirements and any assigned requirements resulting from previous development layers, are used to define a set of technical requirements for the system. The technical requirements guide the solution development process, where the logical and physical aspects of the system are analysed and used to derive further requirements and ultimately a design solution for that system [124]. This design solution specifies re-

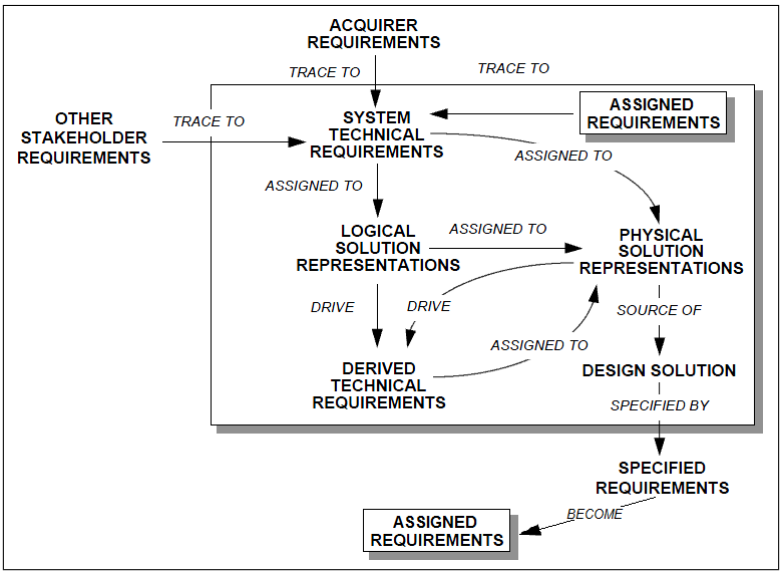



Figure 4.5: EIA-632 requirements relationships [122].

quirements which will become assigned requirements for the next level of subsystem development.

It is important to realise that the standard does not specify the methods or tools a developer should use to implement the process. Instead, this decision is dependent on the the developer and the internal policies of the organization within which he works. The object-oriented model development method proposed in this Thesis would be implemented in the system design stage shown in Figure 4.4. Specifically, as a means of transitioning from the technical requirements to the design solution, while feeding results on for production, analysis and management decisions.

### 4.3.2 ISO/IEC 15288 Standard

Quoting from the introduction of the 15288 standard documentation, its purpose can be briefly summarised as follows [123]:

> "The purpose of this International Standard is to provide a defined set of processes to facilitate communication among acquirers, suppliers and other stakeholders in the life cycle of a system."

Further, ISO/IEC 15288 defines a system as "An integrated composite that consists of one or more of the processes, hardware, software, facilities and people, that

provides a capability to satisfy a stated need or objective" [123].

These standards provide a slightly higher level of abstraction than EIA-632, in that they focus on describing processes within the system's life cycle as a hierarchy of systems while referring to other applicable standards for further development. For example, ISO/IEC 12207 [125] is used for software component development and IEEE 1471 [126] is used for developing architectural descriptions.

The system life cycle processes defined by this standard are arranged into four groups: Agreement processes, Organizational Project-enabling processes, Project processes and Technical processes. Within these groups 25 processes are defined which are to be applied as required within an organization's life cycle model. The life cycle model to be used is not defined by this standard and is to be selected based on the organization's needs.

Typical life cycles can be of the form presented in Section 4.2.2. Further information on life cycle management and the use of a life cycle model for systems in the context of ISO/IEC 15288 can be found in IEEE Standard 24748-1-2011 [127]. In [127], an example is made of a six stage life cycle as shown in Figure 4.6. It is noteworthy that though the stages are shown sequentially, the arrows on the side of each block indicate that it is possible to move forward or backward to a non-adjacent block.

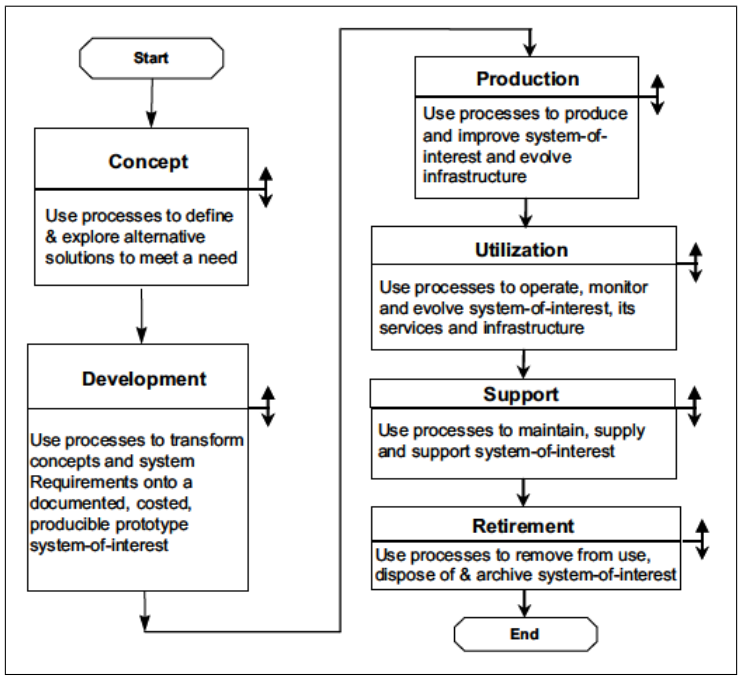



Figure 4.6: Example system life cycle model [127].

Continuing with this example, if this life cycle were to be applied to the development of a HEV powertrain, the proposed object-oriented modelling method could form part of the processes for either the Concept or the Development stages. If new technologies or concepts are to be compared and assessed, the modelling activity forms part of the concept stage. Ultimately the executable models will be used to check the feasibility of the technologies and determine the technical requirements for the development stage. Once technologies have been chosen and the requirements sufficiently refined, the modelling activity forms part of the development stage. In this case the chosen design solution is transformed into a prototype in the form of a complete powertrain model. Also system diagrams, interface specifications and design documentation are produced, while validation and testing is used to further refine stakeholder and user requirements.

## 4.4 Proposed Modelling Method for HEV Development

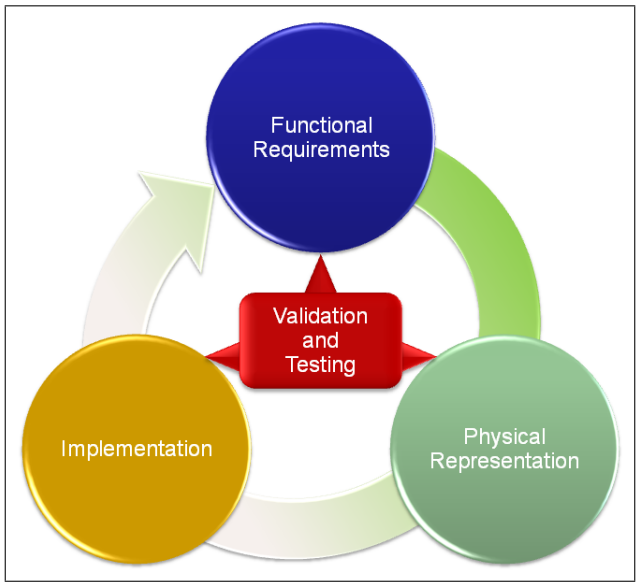



Figure 4.7: Iterative stages of the modelling method.

Just as objected-oriented analysis and design methods are used for building software systems, so a parallel can be made with the method used for developing models using a multi-domain component-oriented modelling language. In fact, Fishwick [96] argues that it is precisely in this arena that the properties of software and

systems engineering converge due to the ties between programming and physical systems modelling.

As stated in Section 4.1, the author proposes using a software and systems engineering approach in order to define the iterative method to be followed when developing object-oriented powertrain models for HEVs. Figure 4.7 illustrates the three main iterative stages of this method along with the validation and testing loop which provides feedback for those three stages.

The remaining sections of this chapter describe the rationale for each stage of this process, with particular emphasis on the author's suggested stepwise development method for the modelling activities. It must be remembered that this method is intended to be used within the design and development stages of a systems engineering framework such as those described in Section 4.3. This enables the evolution of new and complex HEV designs within a structured set of processes that enforce system wide communication of requirements, while still maintaining design flexibility. An overview of the flow of these steps through the three development stages is shown in Figure 4.8. Each step in this flowchart is in turn a subprocess which is elaborated further in the sections that follow. In order to place Figure 4.8 within the context of the systems engineering standards presented in Section 4.3, it can be regarded as a method of implementing the System Design processes of EIA-632 as shown in Figure 4.4, or for the Technical processes of ISO/IEC 15288 such as architectural design and implementation during which would occur during the Concept and Development stages of the system life cycle.

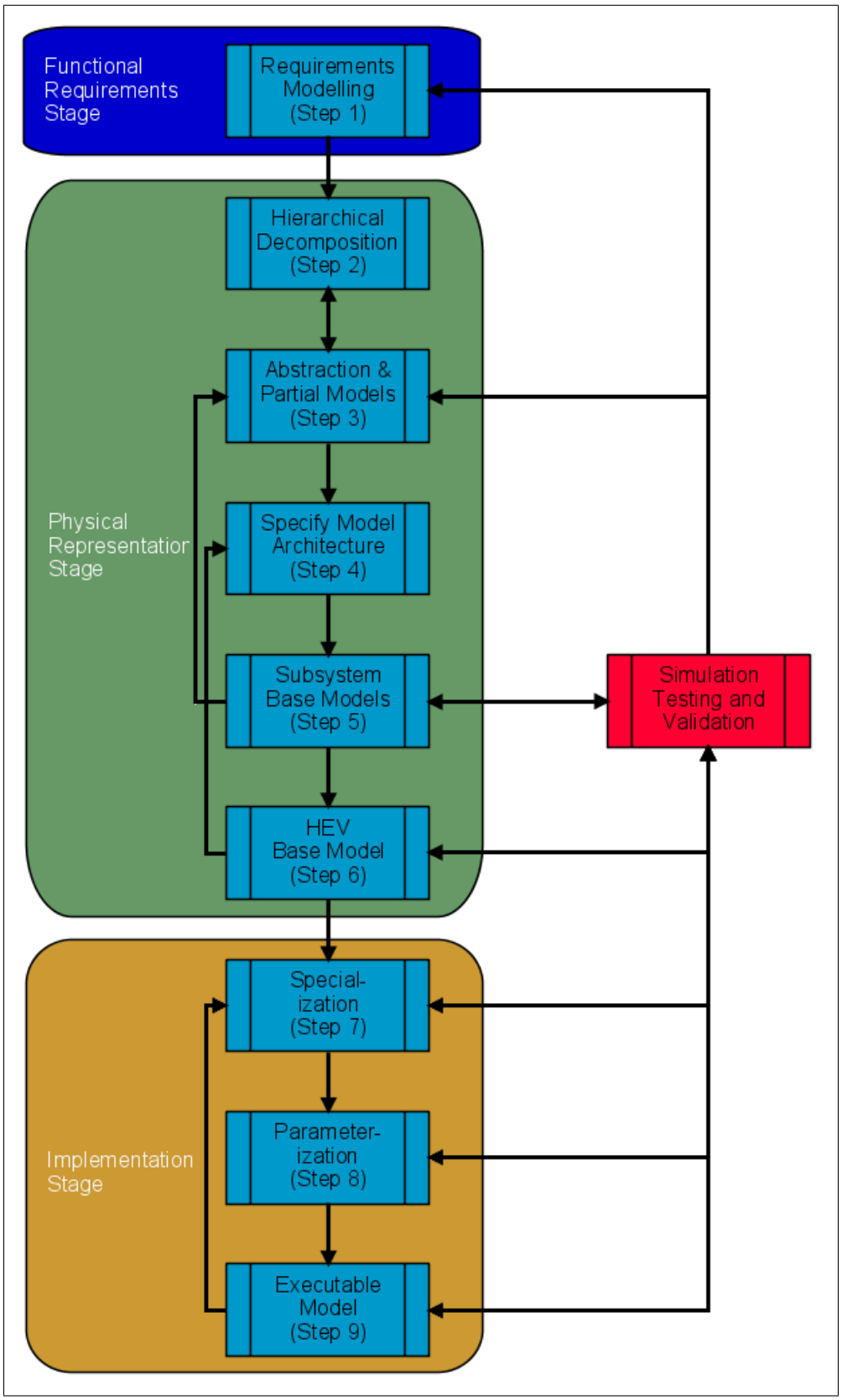



Figure 4.8: Flowchart of the steps in the author's development method.

### 4.4.1 Functional Requirements Stage

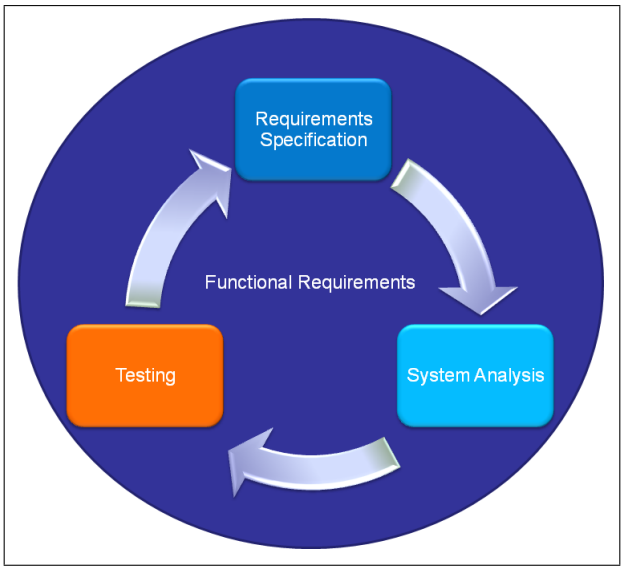



Figure 4.9: Traditional life-cycle development tasks encompassed within the functional requirements stage.

A model should not be overly complicated, yet it also needs to be complex enough to accurately describe the phenomena that are to be studied with that specific model. Williams [128] points out that when dealing with the accuracy and the simplicity of a model, two general principals need to be balanced, namely "Occam's razor" and "requisite variety". The first principle promotes simplicity in seeking for the least complicated explanation and eliminating anything that is not absolutely necessary to achieve the required result. While the second principal essentially states that the ultimate usefulness is limited by the amount of information available. If there is too little detail the results may be imprecise. In both cases, it is the model requirements that dictate where the balancing point for complexity lies.

During the functional requirements stage of the proposed method, the main life-cycle development tasks covered (as listed in Section 4.2.2) are the *requirements specification* and the high level *system analysis* as shown in Figure 4.9. The *testing* task at this stage encompasses both validation that all stakeholder requirements are considered and in agreement, and defining the tests that will determine if the requirements laid out have been met.

In order to determine the requirements and goals of the model, it is imperative that all stakeholders are involved in this process, from management to the domain spe-

cialists and end users. More specifically, this involves the key activity of questioning all parties in order to understand the type of and purpose of model required, as well as all initial constraints and assumptions. Points that need to be considered within the context of HEV development are:

- Why the model is needed,
- What the model should achieve,
- What the intended applications for the model are,
- What level of fidelity is required for subsystem models, and
- What are the major modelling constraints.

A flowchart illustrating the data gathering of the requirements modelling step is shown in Figure 4.10.

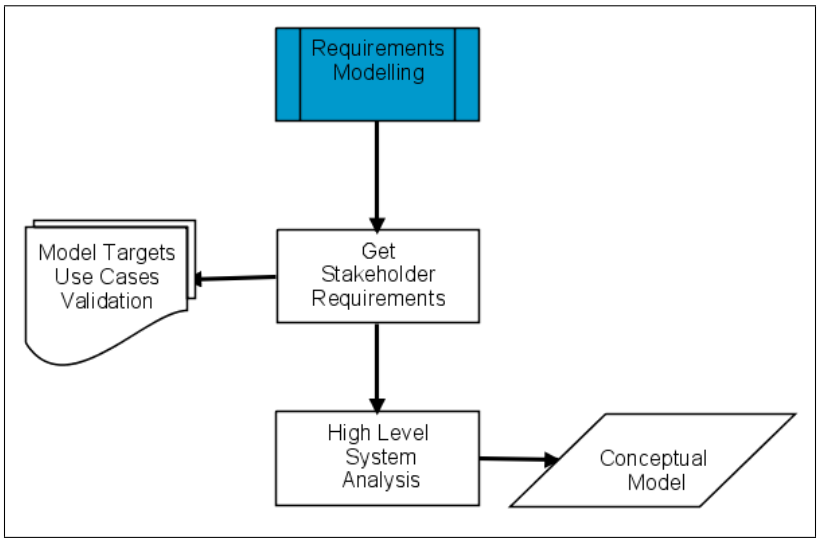



Figure 4.10: Flowchart for the Requirements Modelling Step.

This information is usually collected in requirements specification documents that need to be managed in order to track, update and link these requirements to the developed models as discussed in Chapter 7. Typically this consists of Word documents with tabular data and diagrams used to present the model description, model configuration and model acceptance criteria [129]. Where the description lists the purpose for model, diagrams the hardware and software subsystems and tabulates the specific features to be modelled along with their respective I/O and parameters. The configuration is specific to the modelling environment and describes what model libraries should be used, simulation requirements in terms of speed and step

time, and how the model should be parameterized. Finally the acceptance criteria stipulates the required accuracy for the simulation outputs, the test range for these outputs and the data that they should be validated against.

Marco and Cacciatori [130] highlight the importance of requirements modelling since the functional requirements of a system add value for the end user. Another means of collecting and sharing requirements data from various stakeholders, with the addition of modelling the flow of requirements within a system, is to make use of the UML/SysML use case diagrams. The use cases describe the different tasks that a system or model will be used for, in other words it describes what functions a system will perform in response to different uses. Additionally, these diagrams are linked with the textual requirements descriptions.

The ability of the use case diagram to show the requirements for several alternative decisions, gives developers a more complete picture of the systems functionality. This property helps prevent late changes in requirements due to a misunderstanding of the alternatives, and is particularly useful for HEV controller design where there are multiple possible alternative flows or control decisions [130]. Pressure to get on with modelling and produce tangible results often leads to rushing the requirements stage. However, a lesson from systems engineering shows the costs associated with changing requirements at later stages in the development can be excessively high and often lead to project failure [131].

#### Requirements Modelling (Step 1)

Ultimately this stage of the model design process should clearly establish the scope for the rest of the modelling activity as well as provide the model developers with a conceptual model as shown by the example in Figure 4.11. The conceptual model, in the form of a class diagram showing the high level system breakdown, along with any use cases and associated documents forms the initial phase of the system analysis.

### 4.4.2 Physical Representation Stage

Having determined the type of model that is to be built, the second stage of the modelling method focuses on creating the actual model in a physically representative format. As shown in Figure 4.12, within this stage, the main life-cycle development

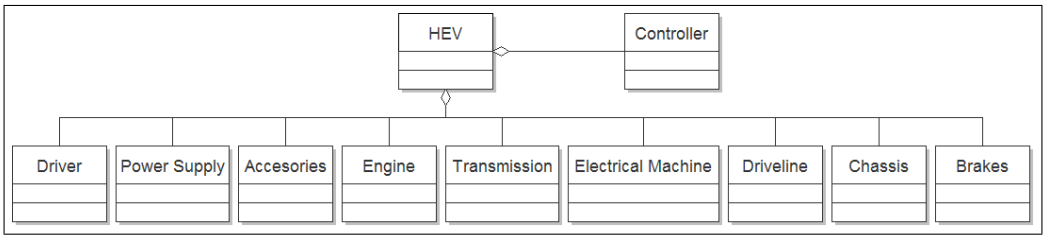



Figure 4.11: Example conceptual model for HEV partitioning.

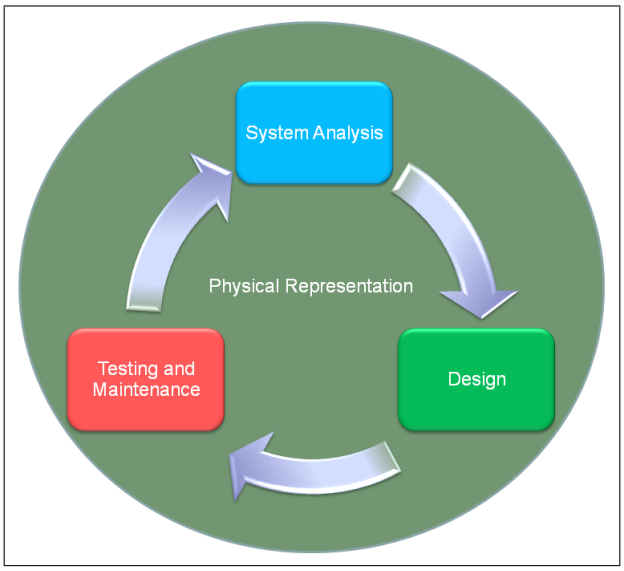



Figure 4.12: Traditional life-cycle development tasks encompassed within the physical representation stage.

tasks performed are: a lower level of *system analysis*, a functional model *design* and finally *testing and maintenance*. Here the testing refers to checking the functionality of the developed model and component classes, and maintenance refers to changes made due to feedback from the other stages in the form of requirements changes or model implementation issues.

This stage can be compared to an engineer creating a blueprint for a prototype design. The activities in this stage are devised to accomplish the following four main tasks:

- the hierarchical decomposition of the system into subsystems and components,
- the abstraction of these subsystems and components in order to determine common base structures,
- defining the interaction between subsystems and components, and
- constructing model libraries.

It should be noted that the remaining stages of the proposed method are developed to take advantage of OOM languages and the features they offer. In particular, features provided by the Modelica language standard and its implementation within the Dymola environment. For this reason the terminology and examples used in the remainder of this report are consistent with those employed within these environments.

The first two steps of this stage can be regarded as furthering the system analysis to a point where initial model design can commence.

### Hierarchical Decomposition (Step 2)

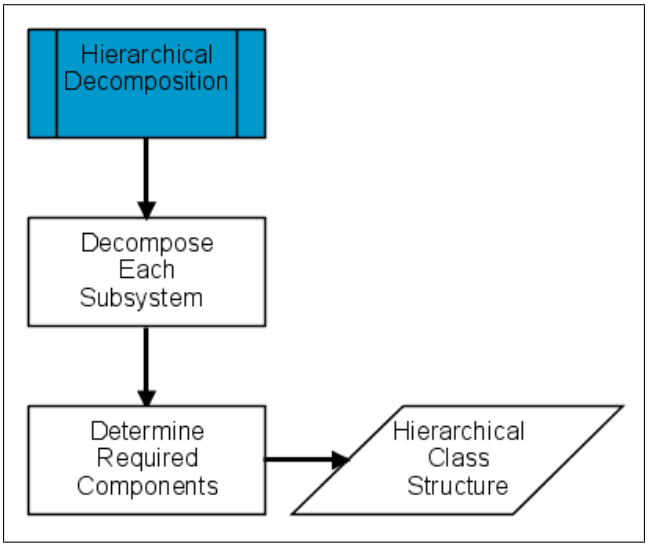



Figure 4.13: Flowchart for the Hierarchical Decomposition Step.

Figure 4.13 depicts the actions taken during the hierarchical decomposition step. In this step the system analysis is taken further by performing a hierarchical decomposition of the powertrain into a logical grouping of subsystems and components that will form the model classes and objects. Defining the necessary objects and classes is relatively simple with respect to the software domain since the actual physical components can be used as guides. As mentioned earlier a model is an approximation of the real-world system, therefore, if the model represents the system being modelled more closely, it is beneficial for both developers and users for building and understanding a model. This is particularly true when modelling large complex multi-domain systems where physically representative models of the subsystems can be aggregated to form the overall model [132].

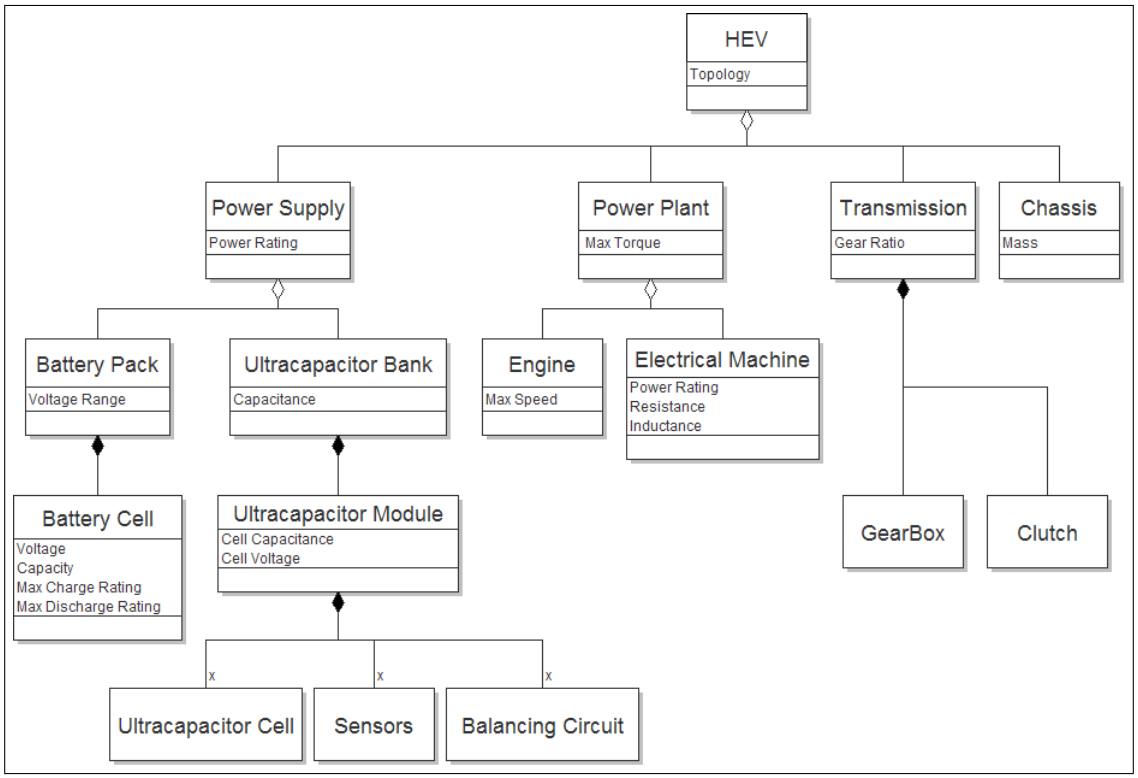



Figure 4.14: Sample subset of a HEV hierarchical decomposition model.

If used together with a suitable system modelling tool such as one based on the SysML, this decomposition can lead to a high level system representation which shows the principle components of the system and how they relate to each other from a hierarchical perspective [105]. An example system decomposition model showing some of the systems of a HEV at different hierarchical levels, is seen in Figure 4.14. The decomposition model can be navigated both horizontally, to examine the various components at a certain level, or vertically in order to explore more detailed decompositions to component level. Further, the availability of such a model can form an invaluable part for managing the communication of data and knowledge at different system and subsystem levels to the necessary stakeholders as mentioned in Section 7.2.1.

The use of decomposition models is also useful for comparing, and possibly automatically generating [133, 134, 135], the many possible architectural topologies that describe how the system and subsystems are interconnected. In this research a flexible architecture similar to the vehicle model architecture (VMA) described in [5, 136] is used. This allows for many configuration changes through replaceable models without making major changes to the high-level architecture. However, the direct

comparison of different logical structures or structural design patterns, for describing the powertrain architecture falls outside the scope of this research and is not explored further in this Thesis. The interested reader is referred to work done by Tummescheit [76] on the use of structural design patterns in mathematical modelling.

### Abstraction and Partial Models (Step 3)

Abstraction is then introduced within the third step of the proposed process, where the higher the level of abstraction, the more generic the definition and therefore the more reusable that abstraction is. However, in order to reduce development effort and allow for larger sections of code or models to be reused, a lower level of abstraction is required. Both experience and iterative design help the developer to decide on an appropriate level of abstraction. An overview of the actions taken during this step is presented in Figure 4.15.

From a practical perspective, the abstraction process is used to create partial models for common vehicle subsystems such as the chassis, wheel, brake, databus and driveline. Referring to the definitions given in Appendix A.1, a partial model is the modelling language equivalent to an abstract class. This abstract or partial model will form the starting point for all variants of that particular model. For instance, chassis models with different levels of fidelity are all constructed by completing the partial chassis model with the relevant dynamic equations.

One particular partial model that should be created after the common or generic vehicle subsystems have been created, is the partial vehicle model. This is a partial model that will be extended to create the base HEV model, later in Step 6, which in turn will be extended to create the required vehicle variants in the Implementation stage as illustrated by the inheritance relationship shown in Figure 4.16.

A sample of the code for the partial model used to build the electrical components with a positive and negative terminal such as resistors, capacitors or voltage sources can be seen in Listing 4.1. At this point, all that needs to be understood from this code is that it is a `partial model` definition describing the relationship between current and voltage variables at the positive and negative pins, and with no functional components within it. By means of the `extends` clause, a resistor component can then inherit the `OnePort` class and define a resistance variable along with an

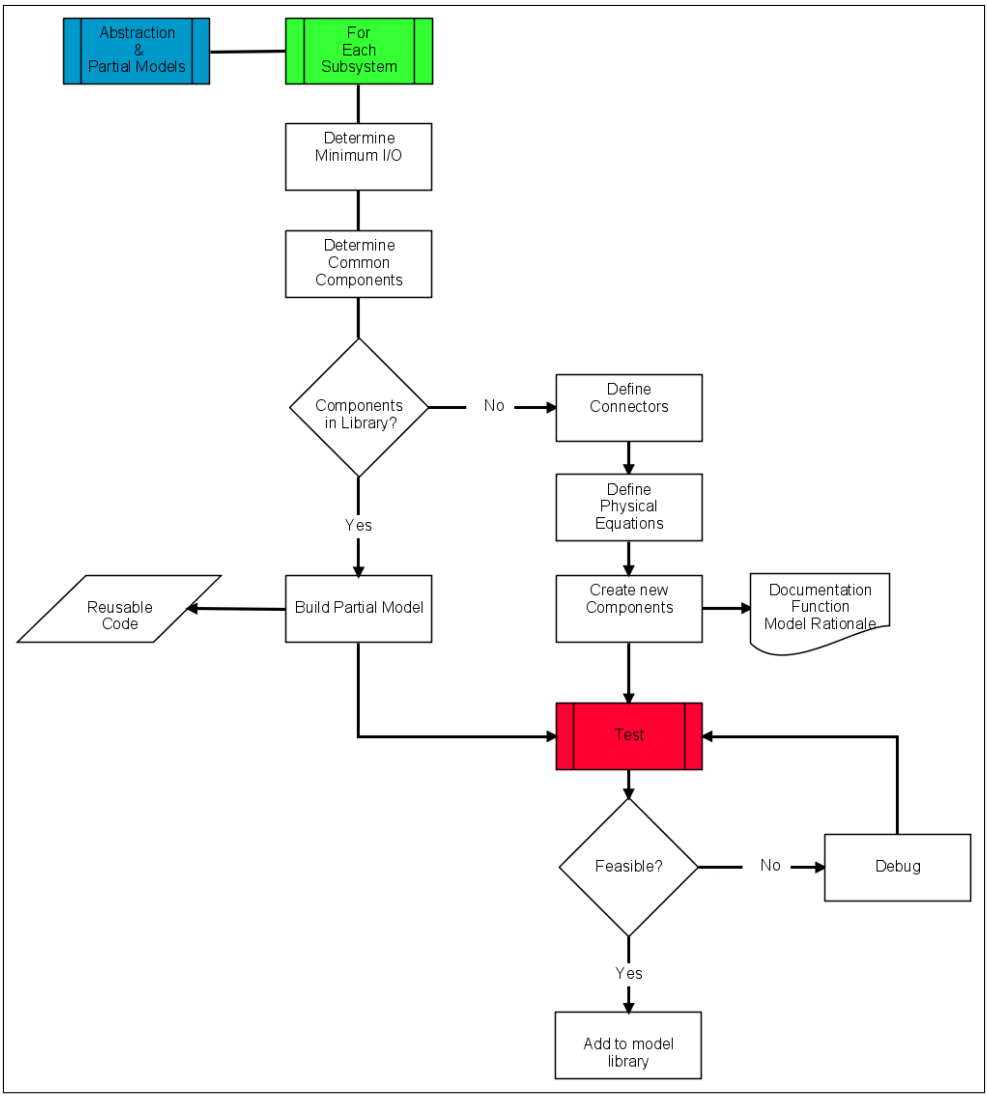



Figure 4.15: Flowchart for the Abstraction and Partial Models Step.

equation relating this variable to the inherited current and voltage variables, thereby describing the physics of the resistor.

The remainder of the steps in the physical representation stage deal with extending or specializing the conventional powertrain components and partial models created in order to develop the hybrid powertrain design.

### Specify Model Architecture (Step 4)

During the "Functional Requirements" stage, the purpose of the model and required level of fidelity for components is decided, but no vehicle topology needs to have been specified. It may be the case that the requirements are to develop models for

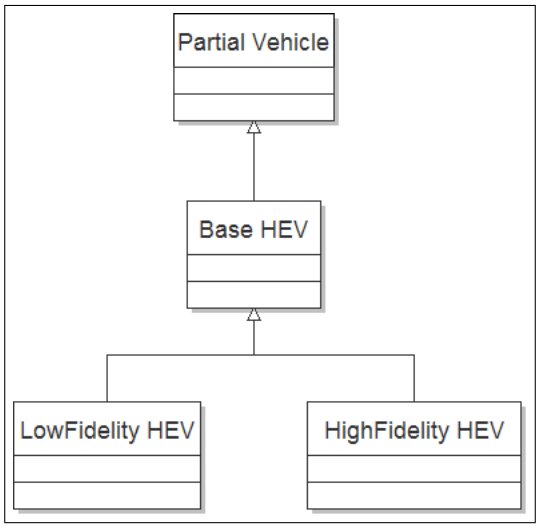



Figure 4.16: Abstraction leads to the following inheritance relationship between the partial, base and final vehicle models.

a specific vehicle type or for investigating several HEV architectures. In either case it is necessary to formalize the vehicle architecture to be modelled at this point. If various powertrain architectures are to be investigated, this step should be iterated or performed in parallel depending on time and resource constraints. At this point, the model developer must consider what additional components and subsystems are required in order to model the chosen type of HEV architecture. This includes new power sources such as ultracapacitors, generators and fuel cells, as well as subsystems for achieving the desired HEV architecture such as drivelines and powersplit devices.

### Subsystem Base Models (Step 5)

Having decided on what needs to be added to the already created components and subsystems, it is now possible to create replaceable base classes as required. As shown in Figure 4.8, this step iterates the modelling process back to Step 3 for the new components and subsystems specific to the selected hybrid vehicle architecture. Additionally, at this step there is a further requirement to create the base models for all subsystems in the HEV powertrain being modelled.

The terms *base model* and *partial model* are often used interchangeably but for the purpose of this work, "*base model*" will refer to a partial model which includes a minimum set of constants (environment) and variables (I/O signals). Base models need not be correctly parameterized and should be used to ensure that all models

```
partial model OnePort "Superclass of Components with two
electrical pins p and n"
  SI.Voltage v "Voltage drop between p and n";
  SI.Current i "Current flowing from p to n";
  Pin p;
  Pin n;
equation
  v = p.v - n.v;
  0 = p.i + n.i;
  i = p.i;
end OnePort;

model Resistor "Ideal linear electrical resistor"
  extends OnePort;
  parameter SI.Resistance R=1 "Resistance";
equation
  R*i = v;
end Resistor;
```

Code Listing 4.1: Example Modelica source code to show usage of a partial model.

built from the base model can meet a minimum set of requirements. Further, the base model will form the the minimum constraint for replaceable models as defined in Section 3.3. Practical examples of replaceable models are given in Chapter 5.

During the three previous steps, hierarchical abstraction was performed, partial models were built, and the specific HEV powertrain architecture being modelled was determined. It is now possible to create base models at this step, whereby the minimum I/O for engine, transmission, power supply, electrical machine and other subsystems must be determined. The main logic for separating this process from the previous steps, is to allow for the generic modelling activities to be performed independently and as early as possible in the design process. This promotes a distributed development process where domain experts can contribute their respective models independently, not having to wait for HEV specific design decisions to be taken by the organisation. Also it is likely that there is a larger knowledge base for the conventional subsystems and as a result, prior model libraries may already have

been developed.

### HEV Base Model (Step 6)

The final step in creating a physical representation of the powertrain system is to extend the partial vehicle model created in Step 3 into a full vehicle base model. The vehicle base model must contain all subsystems for the chosen HEV and should be completely connected to represent the required architecture. This is done by instantiating the base classes, created in Step 5, within the vehicle base model and adding any connections not present in the partial vehicle model such as connections to the HEV specific subsystems. At this point in the design process, the model should represent the complete vehicle system being modelled and it must contain a minimum set of parameters in order to meet the functional requirements.

### Testing and Maintenance

As mentioned at the beginning of this section, testing and maintenance tasks are performed during this stage of the modelling method. However, this task is not manifested as a fixed step in the process but rather forms part of the object-oriented development philosophy as implemented in the design tasks. Specifically during the base model development steps (5 and 6), since with object-oriented modelling each developed class is self-standing and should be tested on completion. In particular, all models can be tested to check that the defined variables within those models are defined using the correct units. This can be done by making use of the `Types` feature mentioned in Section 3.3.1. All variables and parameters should be defined using specific units, such as SI unit, corresponding to their physical quantity. If this is done, the values and units can be exported for use in external models without ambiguity, and a simulation tool like Dymola can automatically check equations for unit compatibility.

Further, recognizing that the base vehicle model meets the minimum functional requirements, testing this model via simulation allows for an initial validation of the system correctness as a whole. The model developer is encouraged to use the simulator in order to provide varying inputs to the model and gain a better understanding of how the outputs correspond to the range of inputs. Since the models

should be built in a non-causal manner, different causalities should also be tested by setting up new simulations with different inputs and outputs defined.

Maintenance is also not a fixed step but should be carried out in a flexible manner as soon as feedback from the requirements and implementation stages action a change, so as to minimize the risk of propagating incorrect models. Managing the risks associated with modelling errors and maintenance are discussed further in Section 7.2.3.

### 4.4.3 Implementation Stage

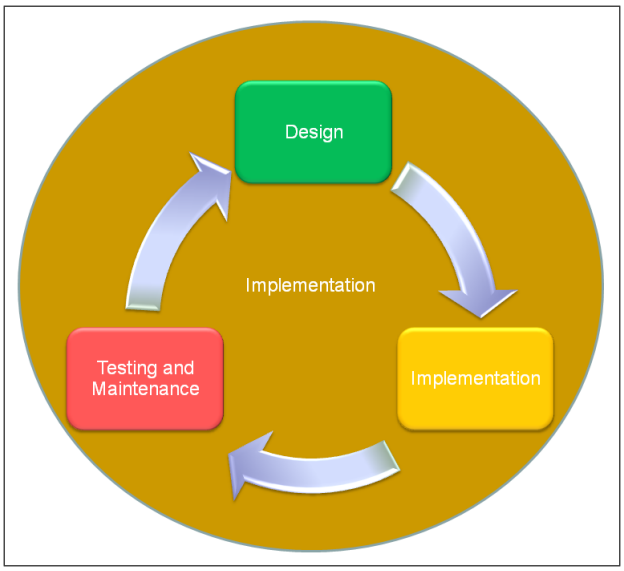



Figure 4.17: Traditional life-cycle development tasks encompassed within the implementation stage

During the implementation stage of the modelling method base model are specialized to the desired level of fidelity and parameterized with meaningful real-world data. The end result of this stage is an executable vehicle model that is more representative of the real vehicle than the base model and can be verified against real-word experimental data. Referring again to the life-cycle development tasks, Figure 4.17 shows that this stage completes the *design* task with specialization and through the *implementation* task, converts the designed model into an executable system model. Again, testing and maintenance of all specialized models forms part of the development process, while validation and verification is performed by simulating the final executable HEV powertrain model.

Following from Step 6 in Figure 4.8, all components and subsystem models at this stage are in a form that completely describes their functionality and interaction. The ability to instantiate all models is necessary in order to put together an executable model that can be simulated. Depending on the requirements set out during the first stage of the method, it is possible that more than one version of a component or subsystem will have to be created. This could be due to differences in parameterization or differences in the required model fidelity. Creation of multiple subsystem variants is done systematically for each subsystem being modelled by iterating Steps 7 and 8 for each variant that needs to be created.

### Specialization (Step 7)

Where possible the base level models created in Step 5 are specialized to the required degree of fidelity for a particular study to be performed on the model. Specializing models by extending from the base models allows for the maximum reuse of previous development effort since the structure, functionality and interface has already been defined in the base model. However, if it is not possible to extend the base model, then the partial model for that subsystem should be used in order to maintain a consistent interface for all model variants. This is done since if the extension of the base model does not change the interface to other models, this provides maximum flexibility in modelling choices since all variants will be able to replace each other. For the case when the base model specialization leads to a change in the interface, this should be used as a maintenance flag indicating that the partial models defining the interface should possibly be revised.

Packages are used to group components and models with similar levels of fidelity. As a means of organizing the multiple variants created within the model library, packages can be nested meaning a package can contain other packages. For this research it was decided to use a package for *energy management* related studies and another package for *high fidelity* related studies. An example of the package structure within the subsystems package is shown in Figure 4.18. Only the ElectricalMachines and Drivelines subsystems are shown for clarity but this structure is followed for each subsystem where a model fidelity change is required. Alternatively, it is possible to create top level packages representing different levels of fidelity and within these packages have all vehicle, subsystem and component models of that particular level of fidelity. With this approach it may be possible to change the

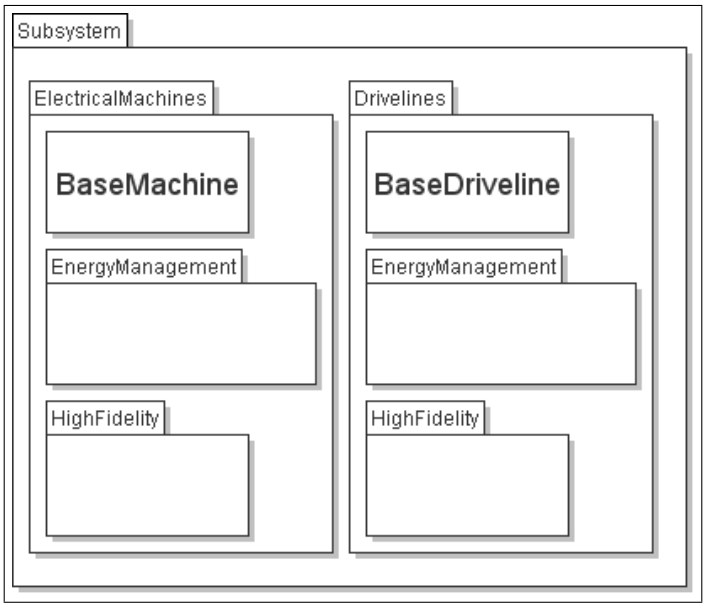



Figure 4.18: Example of Energy Management and High Fidelity packages within the *ElectricalMachines* and *Drivelines* subsystem packages.

fidelity of a simulation model in one step by redeclaring the package but this would mean that the development and changes to any model within one package would have to be reflected in all packages.

Managing such a library may require strict control procedures and it may lead to increased model management effort. Batteh and Tiller [136] point out that a library structure should balance the needs of both the developer of the library and the end user. In [136] the proposed structure for a HEV modelling library is to create implementation packages for each vehicle which contain the specialized and parameterized controllers, subsystems and components required for that specific vehicle. The structure should promote OOM, be easy to navigate, contain parameterized models, separate generic and implementation models [136]. The implications of the overall modelling library structure on the model management are discussed in Chapter 7 Section 7.1.1.

### Parameterization (Step 8)

All components, subsystems and models must be correctly parameterized with meaningful values so that simulation results can be validated against real-world data. Different real-world components such as different types of battery or engine mod-

els have different parameter values and therefore various sets of parameterizations should be stored. This can be done in the following ways:

1. Subsystems can be parameterized at instantiation. This promotes the reuse of the same subsystem models while multiple variants of the complete vehicle model can be created. This approach is often used during initial investigative modelling stages when specific parameters are not known till much later in the design life-cycle.
2. A duplicate class can be created and parameterized with new data representative of a specific real-world counterpart. This is the approach taken in [136] and means that there is an implementation model reflecting each component so that future model developments can be constructed as if by choosing parts from a catalogue. In this case parameterization must occur within the creation of each executable model.
3. Parameter values can be linked to records in a database or data file. It is possible to store different data records for a single model, each record having a different set of parameter values. The benefits of this approach is easier re-parameterization since only the name of the data file needs to be changed. However, this approach requires more effort during model creation in making sure all component parameters link to the database. Model maintenance is made easier since all versions make use of one model, but there is a new additional task of maintaining a database of meaningful and correct parameter data. A further advantage of such a database is that there is a common data repository that can be used and updated by all stakeholders.

**Executable Model (Step 9)**

Finally the full vehicle base model from Step 6 is extended by replacing the base subsystem models with the specialized and parameterized variants created in the previous two steps in order to produce an executable HEV powertrain model with the desired level of fidelity.

**Testing and Validation**

Once each subsystem has been specialized and parameterized, those subsystems are tested again via simulation testing as described in the previous stage. The complete vehicle model is tested in a similar fashion and following this, simulation results are validated against real-world data as supplied by the various stakeholders and as stipulated in the requirements specifications. All specialized subsystems should also be validated against real-world data and the results documented and stored with the model in order to aid the reuse of these models in future model development projects. Finally, all validated subsystem and vehicle models, along with any associated documentation and variable data, must be committed to the model library as the final and latest version. This is done in order to simplify the management of model reuse and maintenance as described in Section 7.1.1.

# Chapter 5

# Case Study 1: LifeCar

This chapter presents a case study on the development of a powertrain model for the *LifeCar* HEV. LifeCar is a lightweight hybrid vehicle powered by a fuel cell in parallel with an ultracapacitor. The driving force of the vehicle is provided by four electrical machines, each directly coupled to a single wheel and connected in parallel to a common voltage bus. Further information regarding the control, modelling and simulation of the LifeCar vehicle can be found in [26, 27, 38, 137].

The primary aim of this case study is to illustrate the use of the author's proposed modelling method for constructing a HEV modelling library and developing powertrain models of different fidelity levels. This case study is a good starting point for testing the proposed method, since it is based on an actual vehicle development effort that had recently taken place, and therefore there was an availability of knowledge both in terms of experience and model data. In the following sections, this case study will be used to demonstrate the application of the proposed modelling method as used in developing a HEV powertrain model library, showing the various modelling stages and steps taken until an executable model meeting a stated set of requirements is achieved. Further, some pitfalls and observations made during this process will be discussed.

## 5.1 Requirements Analysis

As stated previously in Section 4.4, the first stage in the development process is that of determining and communicating the *Functional Requirements*. This is arguably the most crucial stage, as it is considered the root cause of most development problems [138, 139]. That is to say, having a good understanding of who is going to be using the model, how they are going to be using it, and what their expectations from the model will be, is of vital importance in minimizing the possible errors and potentially the number of rebuild iterations required during the subsequent development stages.

The main objective of this study is to design a powertrain library with reusability and extendibility in mind, with the scope of exploring multiple designs reusing most of the underlying components and subsystems. In the case of LifeCar, two variations or instances of the base powertrain model are developed, one based on lower fidelity subsystems for performing energy management studies and another using higher fidelity subsystems for investigating phenomena such as tyre-slip. No real-time simulations were required for this initial investigation of the LifeCar powertrain. Since developing models for real-time and HIL applications form an important part of HEV modelling and development, especially if there is an aim to reduce the costs of prototyping, future work in this area is recommended as mentioned in Section 8.1.

In the first instance a powertrain model is needed that can be executed fast enough to perform drive cycle analyses of parameters such as the ultracapacitor state of charge (SOC) and hydrogen fuel cell efficiency. As mentioned in Section 2.4.4, for this type of model the focus is on determining the energy flow dynamics using quasi-static models and therefore an integration step size in the order of $10\ ms - 100\ ms$ is typically acceptable. Ultimately this type of model will be used to test an energy management control strategy and perform parameter optimisation tests. Therefore the model complexity and frequency range should be low enough to allow for multiple runs over both legislative and real-world drive cycles which have durations over $1000\ s$.

The second powertrain model is to be used to investigate transient torques in the vehicle powertrain, specifically through tip-in and tip-out manoeuvres in order to examine their effect on the acceleration of the vehicle. Additionally, the chassis and tyre subsystem fidelity is increased in order to investigate the effect of tyre-slip on

the longitudinal dynamics of the vehicle. Once again, as mentioned in Section 2.4.4, this type of model is typically used to investigate much higher frequency transient dynamics such as power electronics switching losses and driveline vibrations. Therefore, smaller integration step sizes below 1 $ms$ are required and simulation tests are in the order of several seconds.

As discussed in Section 4.4.1, the requirements analysis phase of any system design process is a lengthy, iterative process involving discussions between all stakeholders, including management, developers and end users. Management of this process is vital in order to complete the project in a timely and efficient manner, since it allows the various domain experts to voice their concerns and should ultimately lead to a consensus between all stakeholders as to the final deliverables of the project. The issue of stakeholder communication is discussed further in Chapter 7, Section 7.2.1.

## 5.2 Physical Modelling

The modelling activity begins with the *Physical Representation* stage of the proposed modelling method. It is in this stage that the requirements are analysed from a design viewpoint and the developer considers how these requirements are to be enforced. The following subsections describe how the modelling steps described in Section 4.4.2 are performed for the LifeCar model. It is worth noting that some of the activities from the proposed method steps have been combined, for instance the abstraction process is described together with the decomposition activity of Step 2. This is done since the actions involved in these two activities follow one another naturally. Also, there is no separate step for specifying the model architecture, since the vehicle topology for this case study is known.

### 5.2.1 Hierarchical Decomposition and Abstraction

In Section 4.4.2, steps 2 and 3 of the method call for decomposing and abstracting the target model design into a set of subsystems and components that can be aggregated to form a complete model description. Initially, the focus is placed on common subsystems that are likely to form part of all vehicle architectures. In the case of the LifeCar modelling exercise, a HEV modelling library is constructed by

firstly including more generic vehicle subsystems such as chassis, wheels and brakes. Following this, the more specific HEV subsystems required to construct the LifeCar powertrain, such as the power supplies and electrical machines, are then added.

Not all the required subsystems are built specifically for the HEV modelling library, since use is made of the freely available VehicleInterfaces [140] library and two commercial libraries, namely the PowerTrain [141] and SmartElectricDrives (SED) [142] libraries.

- The VehicleInterfaces library defines standard interfaces for vehicle and subsystem models with the intention of promoting compatibility between different automotive libraries.
- The PowerTrain library extends from the VehicleInterfaces library and provides components and subsystems for modelling the rotational mechanics of conventional powertrains and specifically focuses on modelling losses in gears and flexible shafts. Additionally, basic components for modelling the vehicle longitudinal dynamics, such as vehicle drag and chassis weight transfer models, are provided.
- The SED library provides components and models for modelling electric drive applications. This includes basic models for various DC power sources such as batteries, ultracapacitors and fuel cells; ideal switching and power balance power electronic converters for AC and DC applications; and various electrical machine models such as induction motors, synchronous motors and permanent magnet DC motors with integrated power electronics and control.

Further information on these libraries, along with the Modelica standard library, can be found in [84, 140, 141, 142].

In theory, since all models in these libraries are based on components from the freely available Modelica library of standard components, which is maintained by the Modelica Association, compatibility between libraries should be ensured. When the Modelica library version is updated, the modelling software is able to automatically update components to their most recent version. However, when proprietary components and models are created or code hiding features are used to protect intellectual property as discussed in Section 7.1.3, this type of automatic model updating is not possible. In this case it is up to the library developer to ensure that the library

models are kept up to date and compatible with the most current version of the modelling software. Since object-oriented programming requires that models or classes be self-standing or encapsulated, it is only the interface or connection to other models that really needs to be compatible. One way of ensuring this compatibility is to enforce the use of standard Modelica connectors. Another means for different simulation packages to ensured model compatibility is by incorporating standards such as the Functional Mock-up Interface (FMI) for Model Exchange [143], as discussed in Chapter 7 Section 7.2.2. The FMI defines a standardized interface for exchanging models as Functional Mock-up Units (FMUs). FMUs contain a minimum of the model equations, in C source or binary format, a separate variable definition file and optional resources such as icons and help files.

The abstraction process calls for breaking down the powertrain into its constituent subsystems, while trying to use as many generic subsystems from conventional ICE powertrain models as possible. This is because the models for conventional ICE vehicle subsystems are already tried, tested and reused in several designs. Thereby increasing design flexibility and reducing both the model validation time and the risk of modelling error through model reuse [144, 145]. For example, when developing an prototype HEV model, a chassis model that has been developed, tested and validated for a specific existing vehicle platform can be reused in a HEV model with new paramterization. This re-parameterized model will then require only a reduced amount of testing since its functionality has already been established.

Figure 5.1 shows the subsystem level abstraction of the HEV powertrain subsystems in the form of a hierarchical aggregation diagram. For each subsystem where there is no existing library model, such as those unique to the particular HEV design, the subsystem must be further abstracted into common components. Once identified, these subsystem models and their constituent component models are used to build a library of partial models. These partial models are, in turn, reused through inheritance and extension to build base models and several complete specialized and executable models. In the case of LifeCar, it was possible to make use of existing chassis and brake models from the conventional vehicle powertrain library. The HEV specific subsystems identified are those of electrical machines, power supplies and the driveline subsystem. The driveline model is specific to the LifeCar architecture since the vehicle does not make use of a differential and each wheel is individually driven by a separate electrical machine.

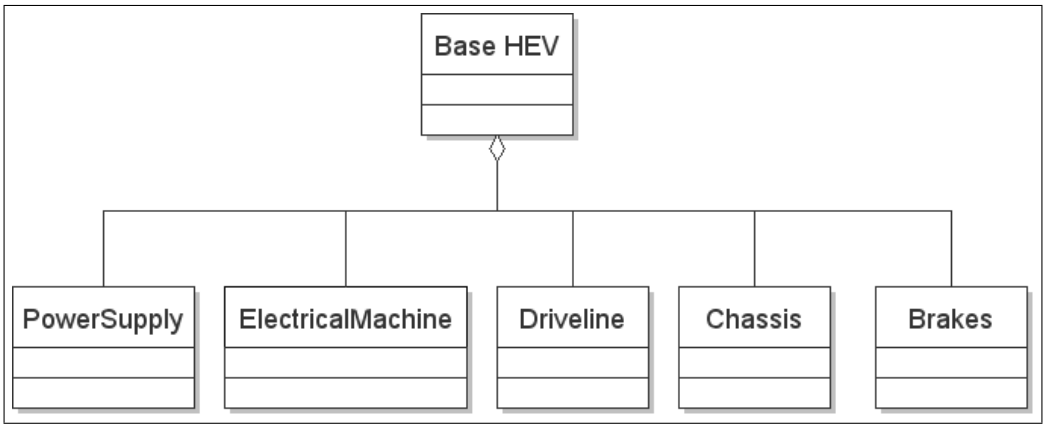



Figure 5.1: Class aggregation diagram showing subsystem abstraction of HEV powertrain.

In terms of connecting the identified subsystems to each other, no new connection types need to be defined. With reference to the model structuring features listed in Section 3.3.1, the predefined Modelica connection types used are described as follows:

1. Flanges (rotational) – for connecting rotational mechanical components such as shafts and gears. This connector defines a variable of type "torque" as the flow variable that will sum to zero at this connection, and a variable of type "angle" as the across or effort that will be equal at this connection. Torques of all components with a joined flange connector will then be summed to zero during translation and the rotation angle for each component will be equal at that connection point. Further the absolute angular velocity at the flange can be determined by differentiating the rotation angle.
2. Flanges (translational) – for connecting translational mechanical components such as masses and springs. This connector defines a variable of type "force" as the flow variable that will sum to zero at this connection, and a variable of type "position" as the across or effort that will be equal at this connection. Forces of all components with a joined flange connector will then be summed to zero and the position for each component will be equal at that connection point. Further the longitudinal velocity at the flange can be determined by differentiating the absolute position of the flange.
3. Pins – for connecting electrical components. This connector defines a variable of type "current" as the flow variable that will sum to zero at this connection, and a variable of type "voltage" as the across or effort that will be equal

at this connection. This ensures that Kirchhoff's current law can then be automatically applied at each electrical node and that the electrical potential is the same for each connected component at that point.

4. Input and Output – these are causal connectors used for defining continuous input or output signals. Typically used for logic and mathematical blocks where the signals are either real, integer or boolean. In this case a connector is defined as being either for input or for output and like connectors cannot be connected to each other. In other words, an input connector can only connect to an output connector and not another input.
5. SignalBus (or controlBus) – for passing various signal types between subsystems. This connector does not model a physical connection and thus does not equate any variables. The controlBus connection is used to allow easy access to all sensor and actuator signals used within the subsystem models and also to simplify the process of defining inputs and outputs when exporting the model to an external environment. Signals are placed on and taken off the controlBus by means of the causal input and output connectors. Further this connector is hierarchical and can therefore contain sub controlBus connectors, for example the power supply subsystem model may contain a controlBus called "powerSupplyBus" which is connected to the top level controlBus. Preserving this hierarchy makes it easier to distinguish between the different signals that are placed on the bus and helps prevent name clashes between similar sensor signals.

Making use of a standard set of defined connectors in all developed models ensures that it is not possible to incorrectly connect subsystem models since connector definitions must match in order to equate the variables. This variable type checking can be performed by the development environment without the need to translate or compile the model. Additionally, the type definitions can specify the units for the connection variables, which enables automated consistency checking in the defined model equations that these variables are passed to.

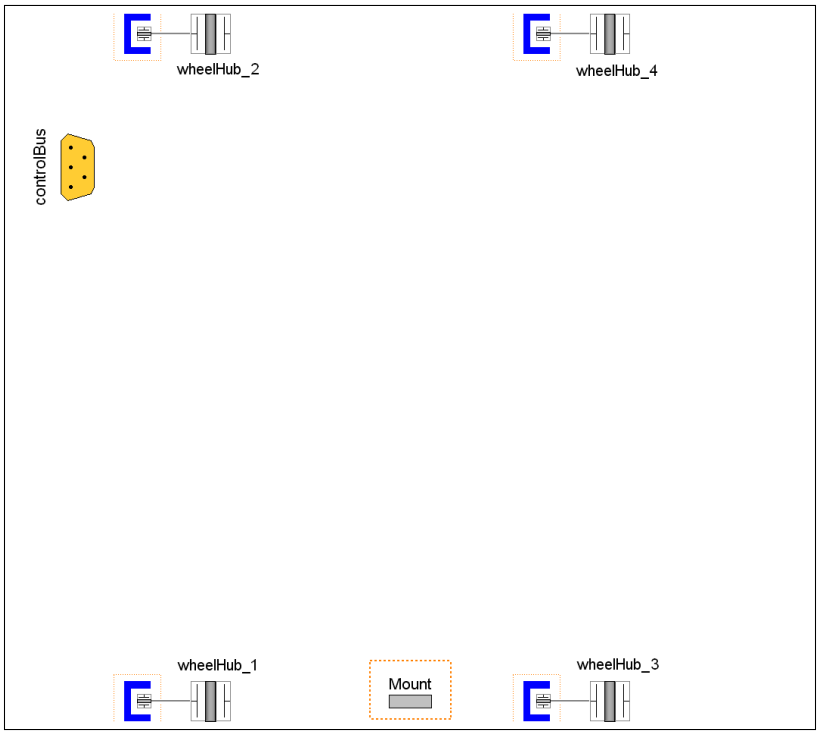
### 5.2.2 Partial Model Design for Common Vehicle Subsystems

Following on from the decomposition and abstraction, the next part of the modelling activity is to create partial models for the defined subsystems according to the process illustrated in Figure 4.15. A partial model needs only to define how that model should interface with other models and not any internal parameters, causal functions or acausal equations. In this way the same partial model can be used to develop various implementations of the same subsystem, for example where each defines a specific level of fidelity.

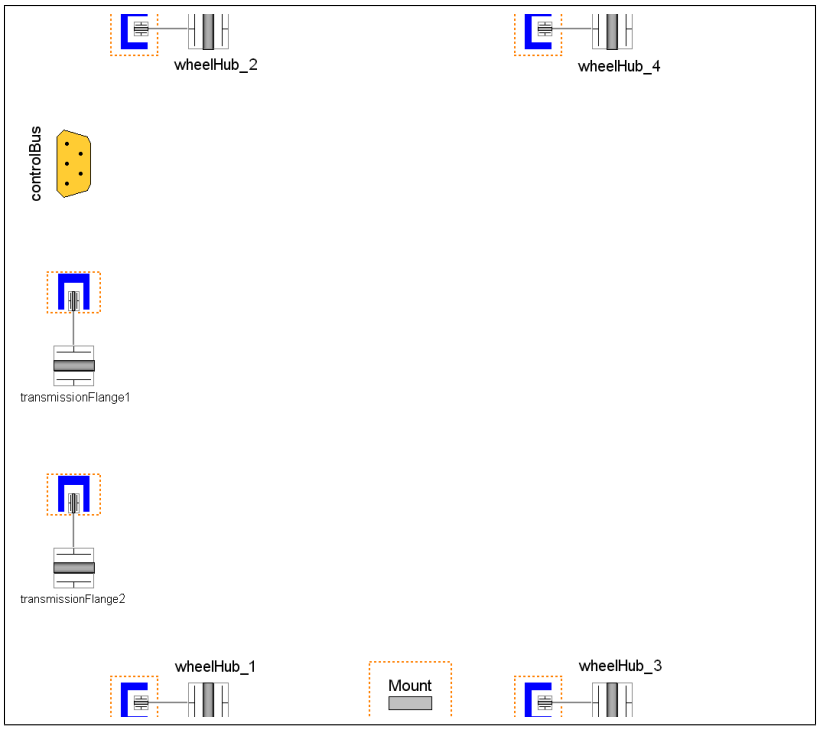
The *driveline*, *chassis* and *brake* subsystem models all make use of a common partial model for a two axle vehicle that defines connections to four wheel hubs via flange connectors and a data sharing connection via an empty controlBus connector as shown in Figure 5.2(a). Since all the available driveline models in the conventional vehicle libraries make use of a single torque input and do not allow for individually powered wheels, it was necessary to create a new partial driveline model specific to the LifeCar vehicle. This is done by inheriting the two axle model and adding two additional flanges to allow for dual torque input connections as shown by the transmissionFlanges in Figure 5.2(b).

Even though the LifeCar requirements are for four driven wheels, a two input partial model rather than a four input partial model was chosen, as this provides more design flexibility. For example if it was decided to investigate a different HEV architecture such as a "through-the-road" hybrid with individually powered front wheels and a separately powered rear axle with a conventional driveline and differential. Four individually driven wheels can be achieved by using two of these dual input driveline models in parallel or extending the two input model with a further two flanges. Further, the dual input model can be used for developing partial two wheel drive powertrain architectures which can then be inherited as the base model for developing a four wheel drive powertrain.

Combining the common partial subsystems into a partial vehicle model is important as this forms the starting point for all future powertrain designs. The partial vehicle model is constructed using partial subsystem models, with each subsystem being defined as `replaceable`. In this way any subsystem implementation designed at a later stage can be used to complete the final powertrain model. Figure 5.3 shows




(a) Partial model for two axle vehicle




(b) LifeCar driveline partial model

Figure 5.2: Partial models for (a) two axle vehicle subsystems and (b) the dual input driveline subsystem for LifeCar.

the partial vehicle model, where the red "NO" symbol (⊘) on each subsystem model simply indicates that it too is a partial model and not executable.

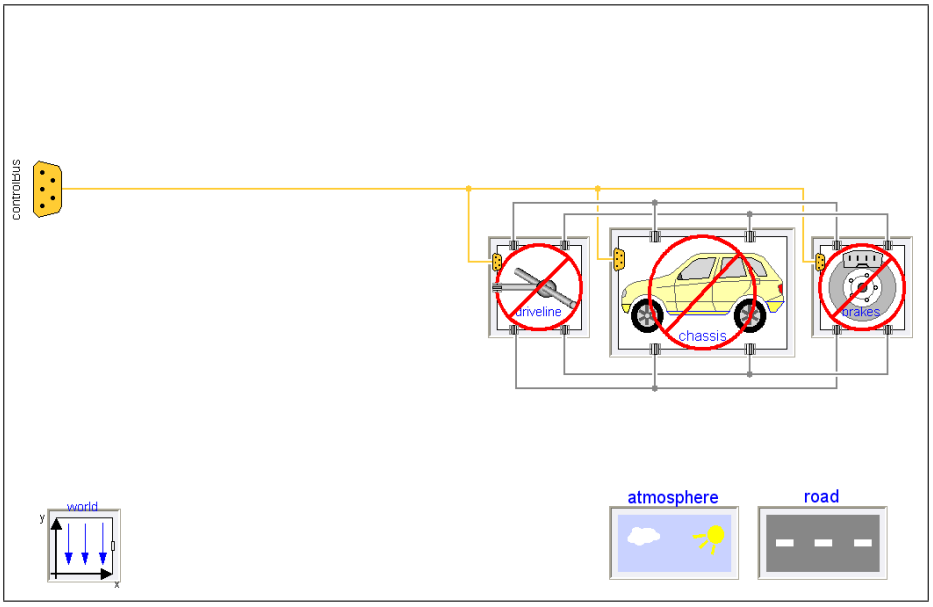



Figure 5.3: Partial vehicle model

This partial vehicle model essentially follows the example set in the VEHICLEINTERFACES library with the exception of a driver environment. It was decided to omit a driver environment from the partial vehicle model in order to allow for fully functional plant models to be developed independently from driver and other high level controller models. Separating the physical plant design from the controller design allows the development of base level models to focus on physical functionality so that initial concept studies can take place using existing or modified controller models. Further, if the plant model is to be exported for Simulink based controller tests or HIL tests, a plant model that can be easily separated from the control functions provided by the external tools is preferable.

Additionally the three models *world*, *road* and *atmosphere* are used for defining typical constants that may be required for the physics based calculations used in the completed models. For the purposes of this work, the *world* model defines a coordinate system and gravitational constant necessary for determining resultant forces affecting the longitudinal vehicle dynamics; the *road* model defines the road friction coefficient that can be used with higher fidelity tyre models; and the *atmosphere* model defines the temperature, humidity, ambient pressure and wind velocity that are used with car resistance models in order to calculate drag forces and with

high fidelity thermal loss models.

The relevant code that describes the partial vehicle model is shown in Listing 5.1. Here it can be seen that the model class is defined using the keyword `partial` signifying that this model is an incomplete model and cannot be executed, it can only be used as a building block for other models through inheritance. Also all instances of the subsystem within this partial model are defined using the keyword `replaceable` so that they can be easily replaced with another version of the required fidelity when the base model and executable models are constructed. An instance of a `controlBus` connector is also defined in the partial model so that all subsystem models can connect their `controlBus` connectors and share the data that they place on the bus. Finally the `equation` section of the partial model definition contains only `connect` statements which define the interaction between connected components. In this case, the `controlBus` connections simply mean that any signals connected to the controlBus connectors within the model will share that data on the top level signal bus, and the `wheelHub` connections define the physical relation through the connected rotational flanges of the *driveline*, *chassis* and *brake* subsystem models. In other words, the rotation angle is equal and the torques sum to zero at each `wheelHub` connection point.

```
partial model PartialVehicle_NoDriver
"partial model for 4 wheeled vehicle without driver environment";
  replaceable VehicleInterfaces.Chassis.NoChassis chassis
  "Chassis subsystem";
  replaceable VehicleInterfaces.Drivelines.NoDriveline driveline
  "Driveline subsystem";
  replaceable VehicleInterfaces.Brakes.NoBrakes brakes
  "Brakes subsystem";
  inner replaceable VehicleInterfaces.Roads.FlatRoad road
  "Road model";
  inner replaceable VehicleInterfaces.ConstantAtmosphere atmosphere
  "Atmospheric model";
  inner replaceable Modelica.Mechanics.MultiBody.World world
  "Global co-ordinate system";
  VehicleInterfaces.Interfaces.ControlBus controlBus
  "Control bus connector";
equation
  connect(controlBus, driveline.controlBus);
  connect(controlBus, chassis.controlBus);
  connect(controlBus, brakes.controlBus);
  connect(driveline.wheelHub_1, chassis.wheelHub_1);
  connect(driveline.wheelHub_2, chassis.wheelHub_2);
  connect(driveline.wheelHub_3, chassis.wheelHub_3);
  connect(driveline.wheelHub_4, chassis.wheelHub_4);
  connect(chassis.wheelHub_1, brakes.wheelHub_1);
  connect(chassis.wheelHub_2, brakes.wheelHub_2);
  connect(chassis.wheelHub_3, brakes.wheelHub_3);
  connect(chassis.wheelHub_4, brakes.wheelHub_4);
end PartialVehicle_NoDriver;
```

Code Listing 5.1: Modelica code for partial vehicle model.

### 5.2.3 Partial Model Design for Hybrid Vehicle Subsystems

Recalling that Step 5 in Section 4.4.2 requires a repeat of the partial model development in Step 3 for HEV specific subsystems, this task is performed before any base model development occurs. When there are multiple developers, it is possible to do this activity in parallel with the base model development for the common vehicle subsystems.

In order to complete the powertrain architecture for the LifeCar vehicle, two more subsystems are needed in the HEV model library. Firstly electrical machines are required to drive the vehicle and secondly a power source for these machines is required. Partial models for these subsystems must first be defined before proceeding to implement any base models even if there are models readily available from existing specialized libraries. The reason for this is that the partial model is what ensures compatibility and will form the basis of new models should the available ones not be appropriate.

#### Electrical Machines

The LifeCar architecture includes four axial flux permanent magnet (AFPM) machines. This type of electrical machine has the benefit of being light weight, compact and having a higher torque density and efficiency than equivalent radial flux machines [146]. As mentioned previously, the SED library contains several commercial off-the-shelf (COTS) models of different electrical machines.

However, it was found that this commercial library was not up-to-date with the latest revisions of the other vehicle modelling packages being used. Consequently, the connectors being used by the SED library models were not compatible with those used in the conventional vehicle subsystem models. Figure 5.4 shows the new partial model that was constructed in order to avoid connector compatibility issues between the different component libraries and enable the use of the SED machine models. This new model defines a data bus connection, positive and negative power terminals and an updated version of the torque flange connector that can be connected to the transmission flanges in the driveline model.

Boehm and Abts [147] points out that the user has no control over the performance, functionality and evolution of COTS products and if that functionality is modi-

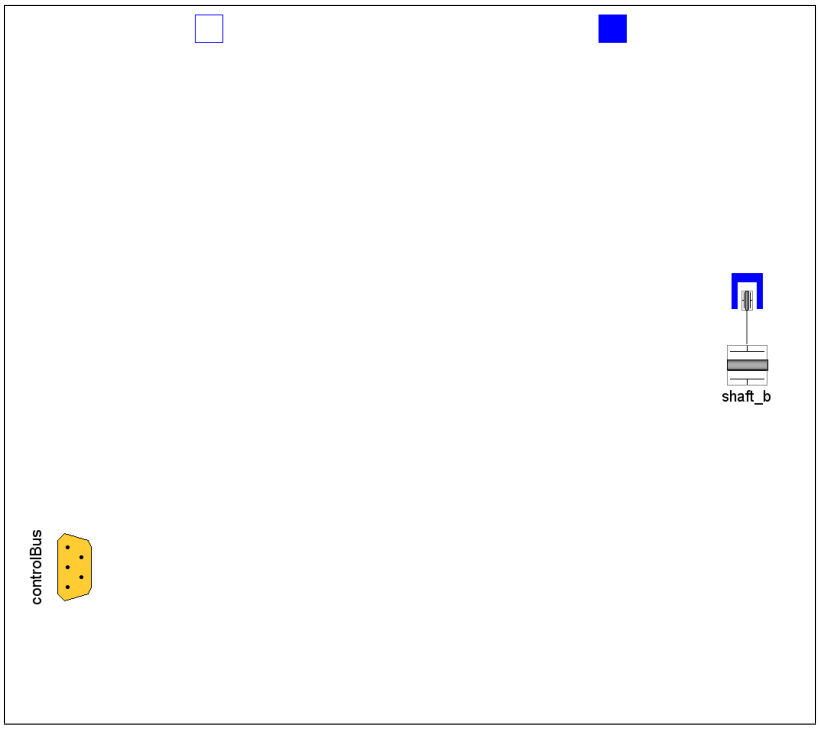



Figure 5.4: Partial model used for encapsulating electrical machine models

fied by the user, the maintenance effort is then transferred to the user. Further, the use of COTS packages requires a flexible development method that allows for risk assessment through prototyping since the capabilities of COTS software are not easily changed to meet requirements and therefore the requirements may need changing [147]. Garlan et al. [148] describe a software development effort where four separate COTS packages are used to produce a new development environment. Ultimately the study shows that the time and effort involved in resolving mismatches between the products caused a two-man half-year project to become a five-man two-year project. Working with several different commercial or free packages involves certain risks and trade-offs which can be both costly and time consuming if not considered carefully [147]. The advantages and disadvantages of using COTS libraries is discussed further in in Section 5.4.

One way to avoid this problem is for developers to make a practice of using a standard and maintained repository for partial models such as the one provided freely by Modelica whenever creating interfaces. Additionally, model developers making use of this library could also submit any new or improved connectors and partial models created for specific application domains so that these can be incorporated into the standard library. Following such a procedure will increase the likelihood

of compatibility between models from libraries developed by third parties. Further, by maintaining and updating this single freely available library it is possible to ensure all developers have access to the latest basic components and interfaces. Such systems exist in the open source community and make use of online development resources such as SourceForge.net to manage and communicate the many changes to the developed code. SourceForge, for example, hosts over 260 000 software projects and has a community of 2.7 million developers [149].

### Electrical Power Supplies

As for the power supply, LifeCar makes use of a hybrid supply consisting of a fuel cell in parallel with an ultracapacitor in order to provide a $400\,V$ voltage bus which supplies the four electrical machines. Although a fuel cell model was available in the existing libraries, it was found impractical to use this model as it required specific parameters which were not known about the fuel cell being used such as partial pressures of the gasses and temperatures. If the only available model has a higher complexity than required, it may be necessary to create a new model. This particular issue can be one of the drawbacks of using models from pre-built libraries.

Availability of pre-parameterized models provides a non specialist user with the opportunity to use a specialized model in order to get some meaningful initial results. This is particularly valuable when designing a complex system with many possible conceptual designs that need to be considered and compared in a timely fashion such as in the field of HEV design. Although there are benefits from the use of shared libraries with readily available functionality, it is also possible that the functionality provided is more than required and leads to performance and usability issues [147]. This can be due to the shared libraries being produced for specialist applications in a particular domain or requiring the user to have advanced knowledge of the particular system. This means that models in these libraries may require specialized knowledge in order to correctly parameterize them, as was the case with the available SED library fuel cell model which was not used in the LifeCar design. In the modelling domain the use of such third party models can lead to unnecessary complexity being included in the model which can lead to stiff models and unreasonably long execution times. Having said this, it is possible to make use of such models if:

1. the library is accompanied by sufficient documentation on the usage of the

model, and more importantly,

2. the library provides parameterized variants of each model that represent actual components that are commonly used. For example a parameterized battery model of a particular type of battery technology such as lithium-ion.

Only a single generic partial model was required in order to construct a modular power supply consisting of fuel cell subsystem and an ultracapacitor subsystem. This is because each subsystem is a also a power supply. The general requirements for an electrical power supply's connections are "Pin" connectors for the positive and negative electrical terminals and a "controlBus" connector for sharing data and parameters that might be required by controllers such as current and voltage readings. The power supply partial model used for creating the fuel cell, ultracapacitor and the hybrid power supply is shown in Figure 5.5.

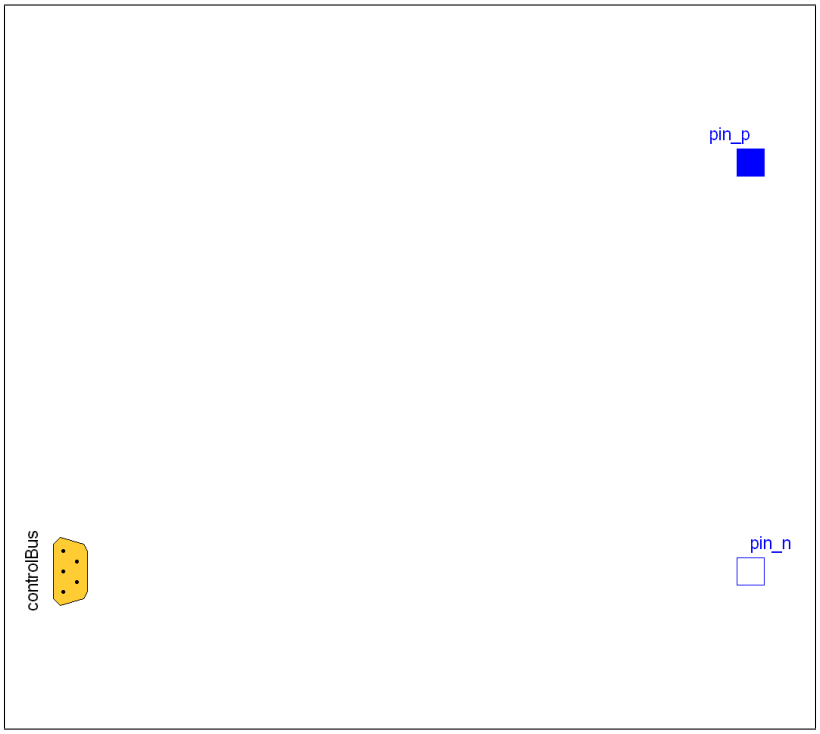



Figure 5.5: Partial model for power supplies

### 5.2.4 Base Model Development: Subsystems and Vehicle

It is now possible, within the defined process, to complete Step 5 by developing all required subsystem base models and then conclude the *Physical Representation*

stage of the modelling method with the construction of the complete HEV base model. Base models are built by either inheriting the partial model and building onto it by instantiating the required components, or by inheriting an existing base model and extending its functionality as could be done to increase the fidelity of the previous base model. This step in the development process is typically iterated several times while developing and testing different subsystem models. For example, during the final implementation stage of the model development, any specialization of models implies a return to this step in the process.

### Mechanical Subsystems

Starting with the mechanical subsystems, it was necessary to select or develop executable models for the *driveline*, *chassis* and *brake* subsystem, remembering that a low fidelity model is suitable at this stage since the primary purpose is to define functionality.

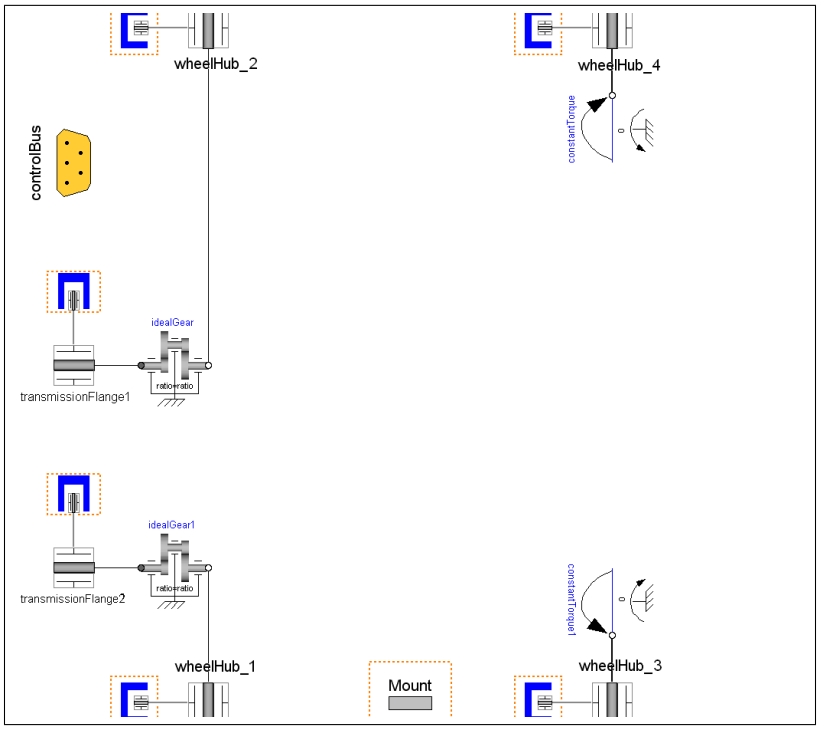



Figure 5.6: Base driveline subsystem model for individually driven front wheels.

To create a working base model of the driveline, an ideal gear is connected between the input flanges and outputs to the wheels. In this case two base models were created, one for front wheel connections as shown in Figure 5.6 and one for the

rear wheels. The addition of the ideal gear makes the base model executable by relating the angle and torque at the input flanges to those at the outputs through the following equations:

$$i_{gear} = \phi_{in}/\phi_{out} \tag{5.1}$$

$$0 = i_{gear}\,\tau_{in} + \tau_{out} \tag{5.2}$$

where: $i_{gear}$ is the gear ratio, $\phi$ is the rotation angle, and $\tau$ is the torque.

The ideal gear does not model inertia, elasticity, damping or backlash but this is suitable for a base model since the main aim is to provide an initial functional base from which further development can occur. During the specialization phase of the design, the fidelity of base models can be increased in order to account for the desired physical attributes. Since separate driveline models are used for the front and rear wheels, a constant zero torque is applied to the unconnected wheel hubs for completeness. Mathematically this has no implication since torques at a common connection sum to zero but this does have the benefit of allowing a potential user of this model to see that it is a complete model and not one with missing components.

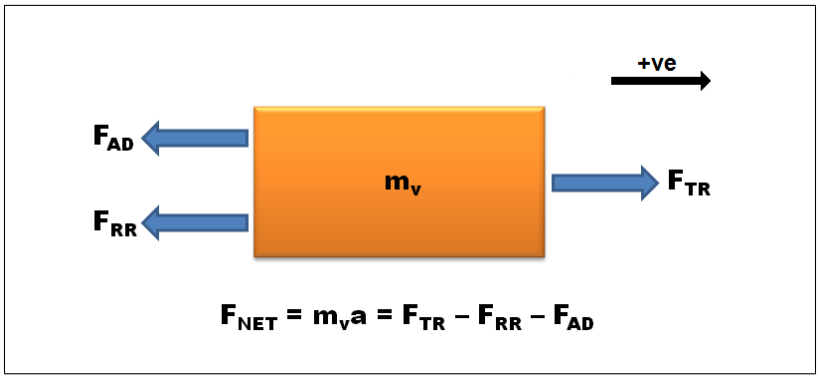



Figure 5.7: Free body diagram showing translational forces acting on vehicle mass.

For the chassis base model, a simple lumped chassis model with no weight transfer effects is selected from the PowerTrain library. A resistance model is used to calculate the one dimensional translational movement of the vehicle mass as shown by the free body diagram in Figure 5.7. The resistance model takes the aerodynamic drag force ($F_{AD}$) and rolling resistance ($F_{RR}$) into account according to equations (5.3)–(5.5). The minimum inputs required by the conventional resistance force model to solve for the net force on the vehicle are; the aerodynamic drag ($C_w$) and

rolling resistance ($\mu_{roll}$) coefficients, vehicle mass ($m_v$) and vehicle frontal area ($A_f$).

$$\mathrm{F_{RR}} = m_v \, g \, \mu_{roll} \tag{5.3}$$

$$\mathrm{F_{AD}} = 0.5 \, \rho_a \, C_w \, A_f \, v^2 \tag{5.4}$$

$$\mathrm{F_{TR}} = \frac{\tau_{TR}}{r_w} \tag{5.5}$$

where: $\rho_a$ is the density of air, $v$ is the vehicle velocity, $\mathrm{F_{TR}}$ is the tractive force, $\mathcal{T}_{\mathrm{TR}}$ is the tractive torque and $r_w$ is the wheel radius.

Additionally the chassis model contains four constant radius wheel models with no tyre slip, which relate the rotational torque and velocity quantities to linear quantities of force and distance via the wheel radius and wheel inertia ($J_w$). Since there is no weight transfer in the chassis model, each wheel receives an equal proportion of the tractive force from the chassis model. During specialization this model can be replaced with models having linear or non-linear tyre slip characteristics as is shown in Section 5.3.3.

Finally a minimal brake model was added that scales the brake pedal position by a combined maximum braking torque for all four wheels and splits this proportionally between the four wheels. Once again this is suitable for the base model as this functionality will be replaced at a later stage depending on the specific needs for the model. For example, the need to maintain the required level of brake balance between front and rear wheels. This model also requires an input from a driver or other controller model to specify the brake pedal position, which needs to be placed on the "controlBus" before the model can be executed.

### Electrical Subsystems

As mentioned earlier, the LifeCar makes use of AFPM machines which are ideal for HEV applications where weight and size can be critical constraining factors. Marco et al. [26] show that these AFPM machines can be modelled as an equivalent torque-controlled DC machine. The following physical equations are used to describe the basic DC machine model:

$$\mathrm{V_a} = k_\phi \, \omega_m + \mathrm{R_a} \, \mathrm{I_a} + \mathrm{L_a} \, \dot{\mathrm{I}}_\mathrm{a} \tag{5.6}$$

$$\tau_m = k_\phi \, \mathrm{I_a} \tag{5.7}$$

$$\mathrm{J_r} \, \dot{\omega}_\mathrm{m} = \tau_m \tag{5.8}$$

where: $k_\phi$ is the torque constant, $\tau_m$ is the machine torque and the remainder of the variables are defined in Table 5.1. The minimum set of parameters required in order to parameterize this DC machine model are those listed in Table 5.1. Notice that the torque constant is not supplied as a parameter input to the model, but rather it is calculated within the model. This is done by considering equation (5.6) under steady state conditions, where $\dot{\mathrm{I}}_a = 0$, giving equation (5.9).

$$k_\phi = \frac{\mathrm{V_a} - \mathrm{R_a}\,\mathrm{I_a}}{\omega_m} \tag{5.9}$$

Table 5.1: Minimum parameters for DC machine model.

| **Variable** | **Description** | **Unit** |
|---|---|---|
| $\mathrm{J_r}$ | rotor's moment of inertia | [kg.m$^2$] |
| $\mathrm{V_a}$ | nominal armature voltage | [V] |
| $\mathrm{I_a}$ | nominal armature current | [A] |
| $\omega_m$ | nominal speed | [rad.s$^{-1}$] |
| $\mathrm{R_a}$ | warm armature resistance | [$\Omega$ ] |
| $\mathrm{L_a}$ | armature inductance | [H] |

Figure 5.8 shows the base model developed by extending the partial model with a basic DC machine. Essentially the partial model is used as a "wrapper" for encapsulating the existing DC machine library models and connecting them to the other subsystem models. The minimum input variables defined by the basic DC machine model are those required to solve equations (5.6)–(5.8) at steady state. These parameter values are typically given on the nameplate of all DC machines and are listed in Table 5.1. The base model does not make use of the "controlBus" connector since the basic machine does not incorporate any power electronic drives or controllers, but the connection is made available in the partial model for the torque and speed controlled machine variants provided in the SED library.

The last subsystem base models to be discussed are the power supply models. The battery and ultracapacitor models used in this study are based on a reduced order equivalent circuit model of LifeCar power supply presented and verified in [137]. Marco and Vaughan [137] show that an accurate high fidelity model of the fuel cell, boost converter and ultracapacitor power supply can be represented as a reduced order resistor-capacitor network. The fidelity of the overall power supply model is

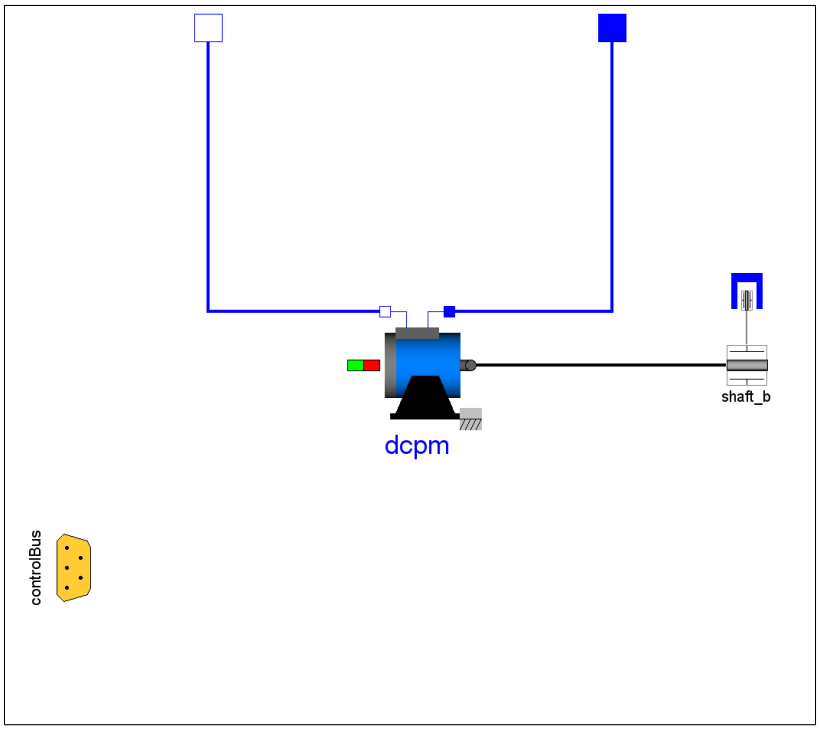



Figure 5.8: Base model for DC machine subsystem

then reduced in order to facilitate controller design by reducing the simulation time required for multiple drive cycle tests.

Firstly a base fuel cell model was built as an equivalent circuit model comprising of a constant voltage source and series resistance. The minimum parameters required for this model were therefore the number of cells, cell voltage and total fuel cell resistance. Next the base ultracapacitor was simply modelled as a capacitor with the option of specifying the starting voltage of the capacitor. Following this a modular power supply base model was created by connecting instances of these fuel cell and ultracapacitor models with a "boost" converter model from the SED library as can be seen in Figure 5.9. The DC/DC "boost" converter from the SED library was used to boost the fuel cell voltage to the ultracapacitor bus voltage by means of a reference voltage $V_{boost}$. This setpoint voltage of the bus is determined in the "voltageBoostLogic" component by means of a lookup table based on the vehicle speed. This is done in order to maximize the benefit of the ultracapacitor by equating its storage energy with the kinetic energy of the vehicle according to equation (5.10). In this way the ultracapacitor will have maximum storage potential when the vehicle speed is high and a braking event is likely, and will be fully charged

when the vehicle speed is low and an acceleration event is likely.

$$\frac{1}{2}\,\mathrm{C}\,\left(\mathrm{V_{max}}-\mathrm{V_{boost}}\right)^2=\frac{1}{2}\,m_v\,v^2$$

$$\mathrm{V_{boost}}=\sqrt{\mathrm{V_{max}^2}-\frac{m_v\,v^2}{C}} \tag{5.10}$$

where: C is the capacitance of the ultracapacitor, $\mathrm{V_{max}}$ is the maximum bus voltage of 400 $V$ and $\mathrm{V_{boost}}$ is the reference voltage for the ultracapacitor.

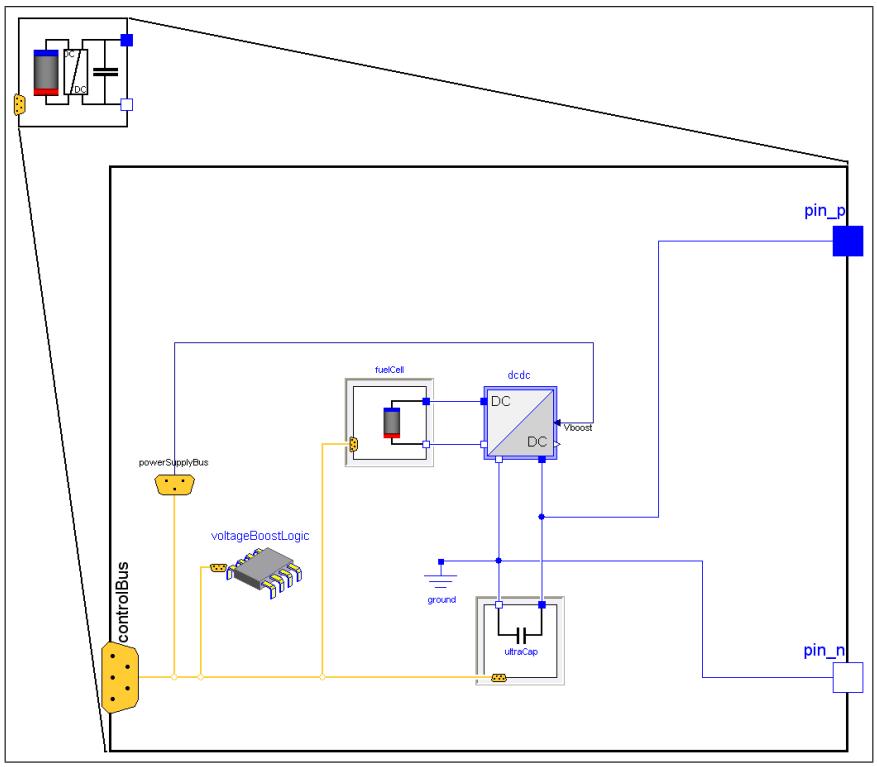



Figure 5.9: Modular power supply model for LifeCar with fuel cell, boost converter and ultracapacitor.

Additionally the fuel cell and ultracapacitor models are made replaceable so as to provide design flexibility in choosing different energy source and energy storage models. Designing the power supply in a modular way, as opposed to a single equivalent circuit model, allows for more reusable subsystem models to be added to the library for future designs. Further, this approach also provides more flexibility to the existing powertrain model for exploring alternate electrical architectures as demonstrated in Chapter 6.

### Complete HEV Powertrain

At this point in the modelling activity the final step of the *Physical Representation* stage described in Section 4.4.2 is reached. The developed HEV modelling library contains enough subsystems to define a base model which represents the complete powertrain architecture for LifeCar. This is done by inheriting the partial vehicle model, replacing the partial models of the common subsystems with base models and then aggregating the new hybrid subsystem by adding instances of the base models as illustrated in Figure 5.10. Specifically, four instances of the base DC machine models and an instance of the hybrid power supply model are added.

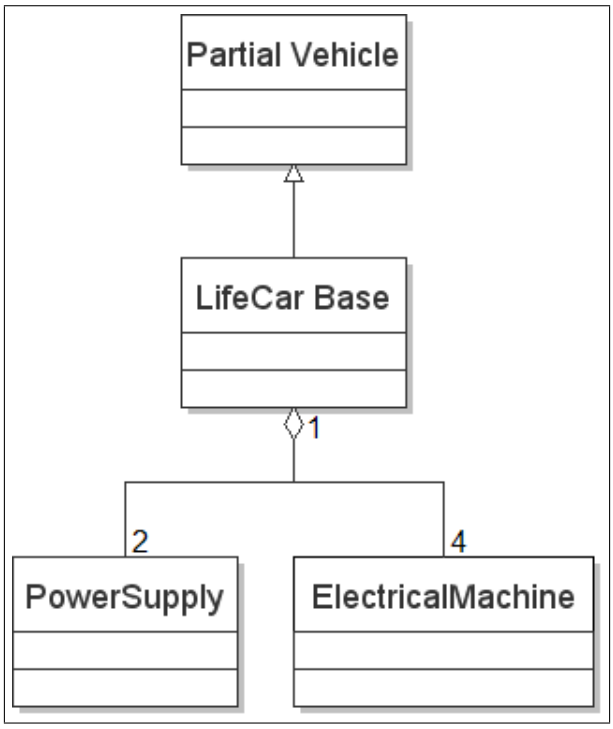



Figure 5.10: Class diagram showing LifeCar base model structure

It is important at this stage to ensure that the newly added components are all made `replaceable` so that the base model can be used as a parent class for further specialization. With reference to Listing 5.2, the line starting with the keyword `extends` shows how the LifeCar base model is formed by inheriting the partial vehicle model and replacing its empty subsystems with base subsystems using the keywords `redeclare replaceable`. The powertrain base model is then completed by instantiating the remaining subsystems as `replaceable` models, namely the electrical machines, a driveline for the front wheels and a powersupply.

```
model LC_BaseModel
"Powertrain base model for LifeCar"
 extends HEV_Modelling.PartialModels.Vehicle.PartialVehicle_NoDriver(
   redeclare replaceable BaseSubsystems.Drivlines...
                         ...Driveline_IRD_Ideal driveline,
   redeclare replaceable PowerTrain.Chassis...
                         ...ResistConventionalRconst chassis,
   redeclare replaceable VehicleInterfaces.Brakes...
                         ...MinimalBrakes brakes);
 replaceable HEV_Modelling.BaseSubsystems.PowerSupplies powerSupply
 "Hybrid supply with Fuel Cell, Boost Converter and Ultracapacitor";
 replaceable BaseSubsystems.Drivlines.Driveline_IFD_Ideal frontWheels
 "Driveline subsystem for front wheels";
 replaceable BaseSubsystems.ElectricalMachines.DCMachine_Plain rearR
 "Permanent magnet DC machine";
 replaceable BaseSubsystems.ElectricalMachines.DCMachine_Plain rearL
 "Permanent magnet DC machine";
 replaceable BaseSubsystems.ElectricalMachines.DCMachine_Plain frontR
 "Permanent magnet DC machine";
 replaceable BaseSubsystems.ElectricalMachines.DCMachine_Plain frontL
 "Permanent magnet DC machine";
equation
 [connect statements]
end LC_BaseModel;
```

Code Listing 5.2: Modelica code for LifeCar powertrain base model.

The final base model for LifeCar is represented in Figure 5.11. As mentioned in Section 5.2.2, a driver model is not included in the base model in order to develop a plant model independently from controller functions. Further, not including a driver model at this stage makes it simpler for powertrain models developed through inheritance of the base model to be used for co-simulation and HIL studies where the external software and hardware will inherently provide the control functions. As is discussed in Section 5.3, the base model will then be extended by adding a driver model in order to create the executable models during the *Implementation* stage of

the modelling method.

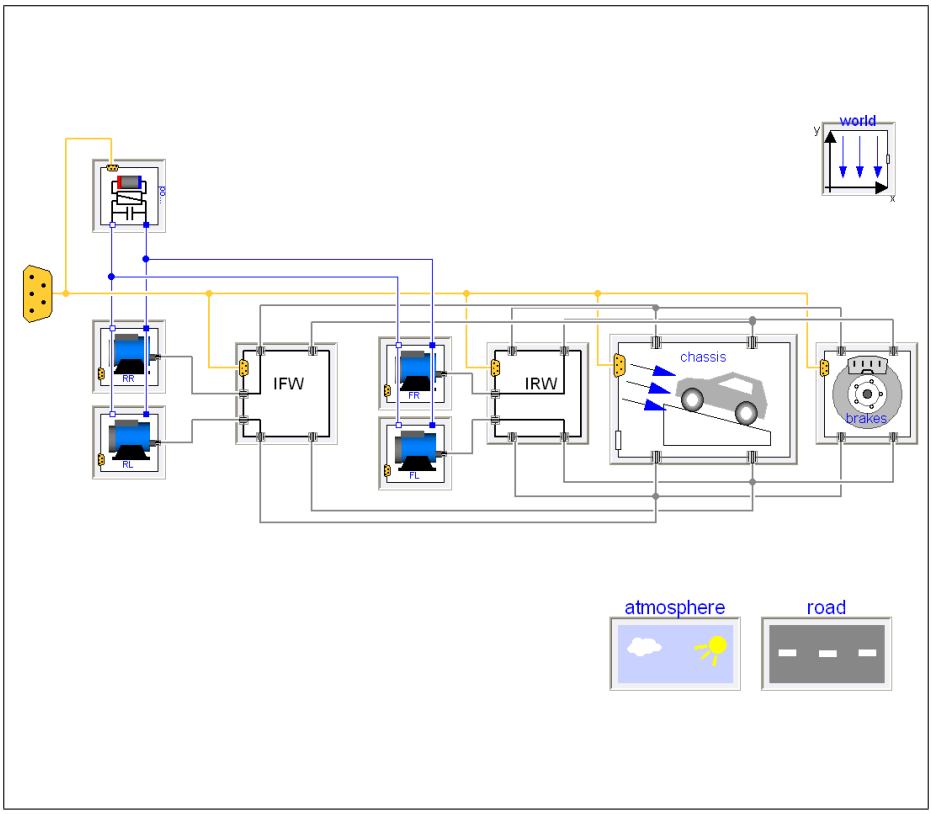



Figure 5.11: Base model for the LifeCar powertrain.

## 5.3 Executable Model

The final stage in the modelling method is the *Implementation* stage. During this stage new subsystem models are created, or existing subsystem models are extended, wherever more detail is required in order to meet the modelling objectives. These new subsystems are stored within the subsystem structure in a separate package denoting their level of complexity. As discussed in Section 4.4.3, there are different ways to configure a model library and it should be convenient for both the developer of the library and the end users. For this case study, an *EnergyManagement* and a *HiFidelity* package were added within the subsystem packages where further specialization was performed, such as in the "Drivelines" and "ElectricalMachines" packages. The same package structure was used again in the "LifeCar" powertrain package for specialized versions of the complete vehicle powertrain model. The structure of the packages within the model library are shown in Figure 5.12. The library

structure was chosen in order to make future use and development more intuitive by enabling users to search for models in a similar fashion to locating a component in a catalogue. In other words the highest level is arranged by subsystems and within each subsystem package are the various specializations of that subsystem. Section 7.1.1 looks at the rational for choosing and managing a library structure in more detail.

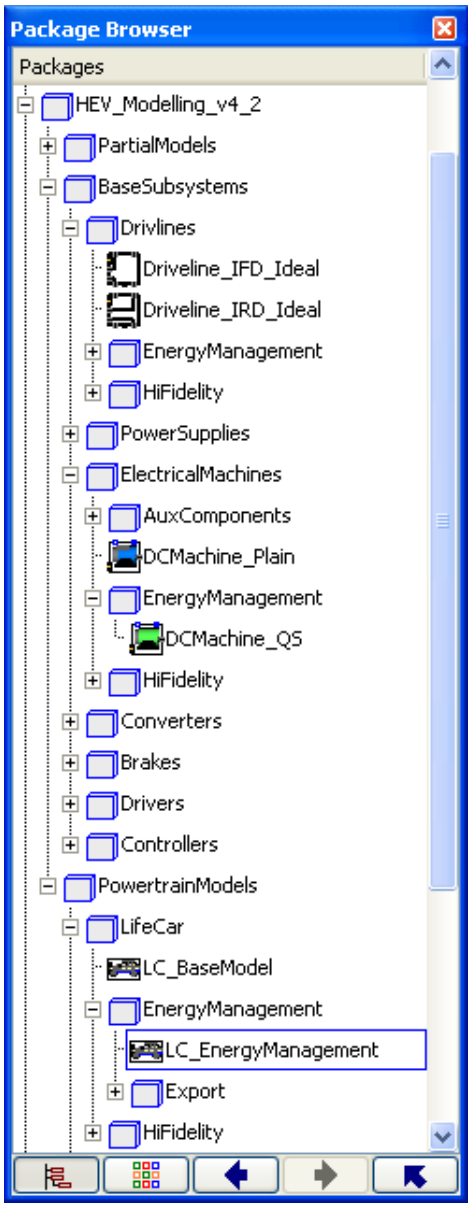



Figure 5.12: Tree list showing package structure within the model library.

### 5.3.1 Energy Management Model Specialization

#### Driveline

Firstly the base driveline models were specialized by replacing the ideal gears, between the transmission and wheel flanges, with gear models from the Modelica library that incorporate mesh efficiency due to friction in the gear teeth and bearing friction losses as shown in Figure 5.13. This allows the user to specify a table of mesh efficiency ($\eta_{mesh}$) and friction torque ($\tau_{friction}$) variables for different input shaft speeds. For a more detailed explanation of how this is implemented in Modelica code please refer to the Modelica Standard Library documentation [150]. These variables are then used to implement the following torque balance equation:

$$-\tau_{out} = i_{gear}\left(\eta_{mesh}\tau_{in} - \tau_{friction}\right) \tag{5.11}$$

where: $i_{gear}$ is the gear ratio.

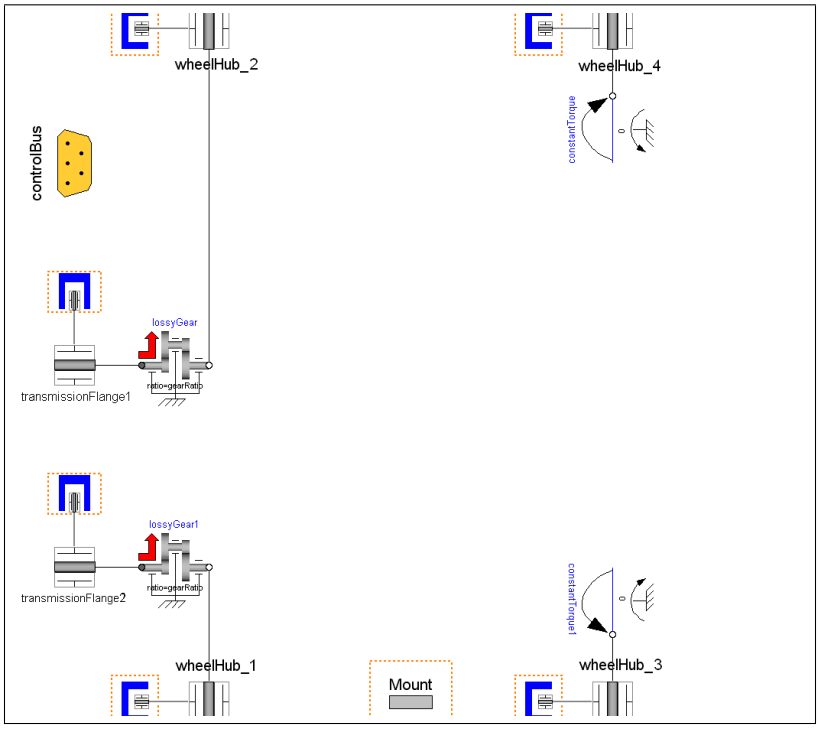



Figure 5.13: Driveline subsystem specialized for energy management.

Since the bearing losses and mesh efficiency for the used gears was not know, the model was parameterized with an overall gearbox efficiency of 90%. The overall efficiency is a commonly cited efficiency value in gearbox catalogues which describes the ratio of output power over input power [151].

### Power Supply

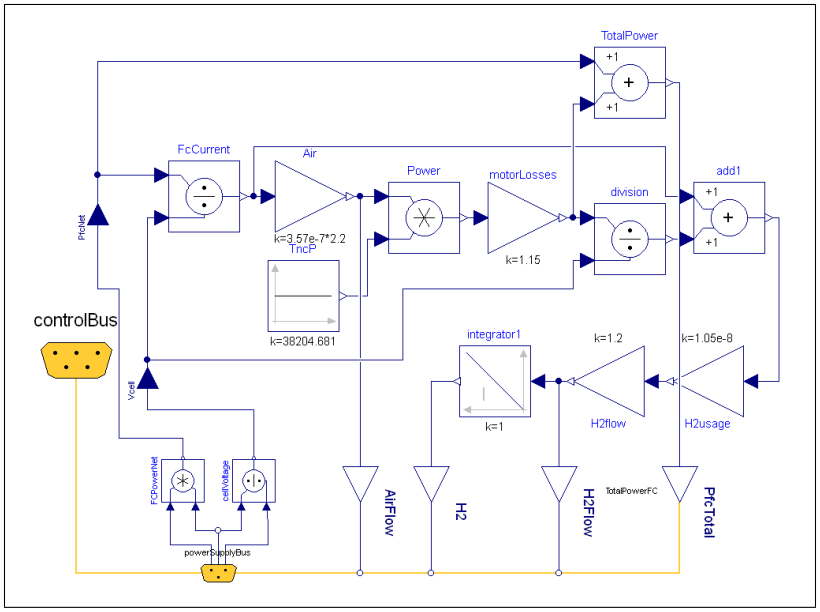



Figure 5.14: Model for calculating the air, hydrogen and total power usage of the fuel cell.

The only change made to the hybrid power supply model was to add the balance of plant model shown in Figure 5.14 within the fuel cell model. The balance of plant in this case refers to the inlet air compressor for the fuel cell which places an increased power demand on the fuel cell. This model takes that additional power demand into account when calculating the hydrogen consumption of the fuel cell. As can be seen in Figure 5.14, the model makes use of the "powerSupplyBus" which is a SignalBus connector used by the power supplies to place shared data onto the common "controlBus". Current and voltage readings from the fuel cell model are taken from the data bus and used to calculate the air mass flow rate ($\mathrm{MF_{air}}$), hydrogen mass flow rate ($\mathrm{MF_{H_2}}$), hydrogen mass ($\mathrm{m_{H_2}}$) and total fuel cell power usage ($\mathrm{P_{fc_{total}}}$) by implementing equations (5.12)–(5.15).

$$\mathrm{MF_{air}} = 3.57 \times 10^{-7}\, \lambda_A \left( \frac{\mathrm{P_{fc_{net}}}}{\mathrm{V_{cell}}} \right) \tag{5.12}$$

$$\mathrm{P_{fc_{total}}} = \mathrm{P_{fc_{net}}} + \mathrm{P_{comp}} \tag{5.13}$$

$$\mathrm{MF_{H_2}} = 1.05 \times 10^{-8} \lambda_H \left( \frac{\mathrm{P_{fc_{total}}}}{\mathrm{V_{cell}}} \right) \tag{5.14}$$

$$\mathrm{m_{H_2}} = \int \mathrm{MF_{H_2}} \tag{5.15}$$

where: $\lambda_A$ and $\lambda_H$ are the stoichiometry scaling factors for air and hydrogen respectively, $P_{comp}$ is the power required to drive the fuel cell compressor and $P_{fc_{net}}$ is the product of the fuel cell current and voltage.

A detailed description of the derivations of these equations can be found in [152].

### Electrical Machines

As mentioned earlier, the SED library was used to provide the electrical machine models. For the energy management application, the simple DC machine in the base model was replaced by a controlled quasi-static DC machine. The quasi-static model neglects transients electrical effects caused by the machines inductances, since inductances have inertial effects on currents, in order to allow for faster simulation [142]. Unfortunately, it was not possible to examine the detailed construction or underlying code of this quasi-static DC machine in order to see how this model simplification was implemented. Once again this is an issue that relates to the use of COTS models and the protecting of intellectual property as discussed in Chapter 7.

This type of model was selected for the energy management specialization since the long term energy consumption over one or more drive cycles is of primary concern. Therefore, it is suitable to ignore the fast electrical transients which have minimal influence on the electrical machines power usage, and allow for faster simulation [153]. Further, this model implements a simplified control strategy based on the physical system equations of the DC machine given in equations (5.6)–(5.8). Therefore this model can be used in the LifeCar powertrain model as a torque controlled device.

### Powertrain

Following from this, the complete executable LifeCar energy management powertrain model shown in Figure 5.15 is developed. This is done by inheriting the base vehicle model shown in Figure 5.11 and replacing the base subsystem models with the newly created energy management subsystems. The chassis and brake models defined in the base model were not replaced as they were deemed suitable for this purpose.

Further additions made to the model in order to make it executable included:

- a proportional-integral controller based driver model capable of providing ac-

celerator and brake signals for predefined drive cycles was selected from the PowerTrain library, and

- a torque reference input for the DC machine controller calculated by scaling the accelerator signal by the maximum machine torque.

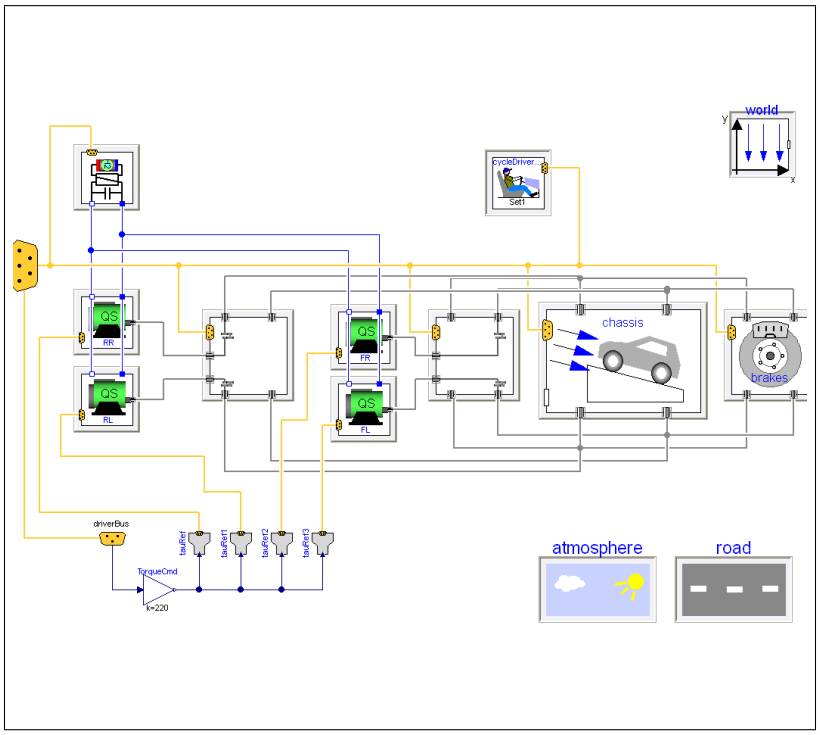



Figure 5.15: Executable energy management model of LifeCar

### 5.3.2 Energy Management Model Validation Tests

As stated in the introduction to this chapter, the main purpose of this case study is to test the modelling method and not to support the LifeCar design activity. In that light, the following validation tests were performed in order to establish that the models created were in fact feasible plant models with reasonable results. This was done in three ways; simulating the model over a drive cycle in the Dymola simulator, porting the powertrain plant model into Simulink to perform a drive cycle test with an existing energy management controller, and finally comparing the Dymola plant results with an existing validated Simulink model of LifeCar. The existing Simulink model was previously validated against experimental data, as described in [26, 38].

**Simulation Tests in Dymola Environment**

The first part of the model validation involves testing the LifeCar energy management model within the Dymola environment in order to asses if the results are mathematically correct and to verify that there are no anomalies being introduced by the model. This is done by checking the simulation output values at chosen points with hand calculations, and visually comparing the overall trend of results with those of the verified Simulink model. Since no energy management controller was implemented within the Dymola environment, the magnitude of the simulation results is not expected to be representative of the actual vehicle. Also since there is no regenerative braking, all machine torque demands are positive. Simulation test runs were performed on the model using European and American legislative driving cycles such as the modal New European Driving Cycle (NEDC) and the transient Urban Dynamometer Driving Schedule (UDDS).

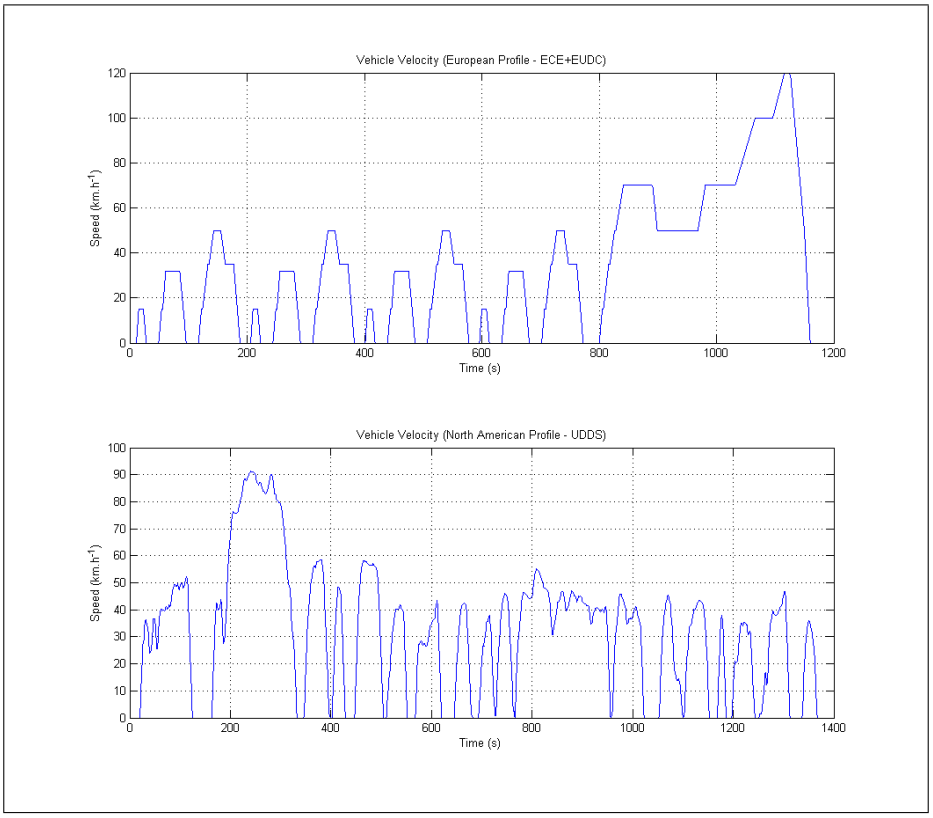



Figure 5.16: European and North American (UDDS) drive cycles used for Dymola based simulation tests

Transient cycles are more representative of actual driving, whereas modal cycles have periods of constant speed and don't represent a realistic driving pattern as can be

seen in Figure 5.16. The NEDC cycle simulates some urban driving with a maximum speed of 50 km/h followed by a high speed period reaching a maximum of 120 km/h while the UDDS cycle represents a transient urban route with many acceleration and deceleration events and a maximum speed of 91 km/h. A second more aggressive North American driving cycle is given by the high speed and high acceleration US06 cycle with an average speed of 78 km/h and a top speed of 130 km/h.

Simulation results for the NEDC test are shown in Figure 5.17. The results show that the vehicle follows the drive cycle speed and in doing so uses 43g of hydrogen while discharging the ultracapacitor between 400 V and 331 V. Any recharging of the ultracapacitor is due to the fuel cell providing enough current to bring the bus voltage back to the setpoint.

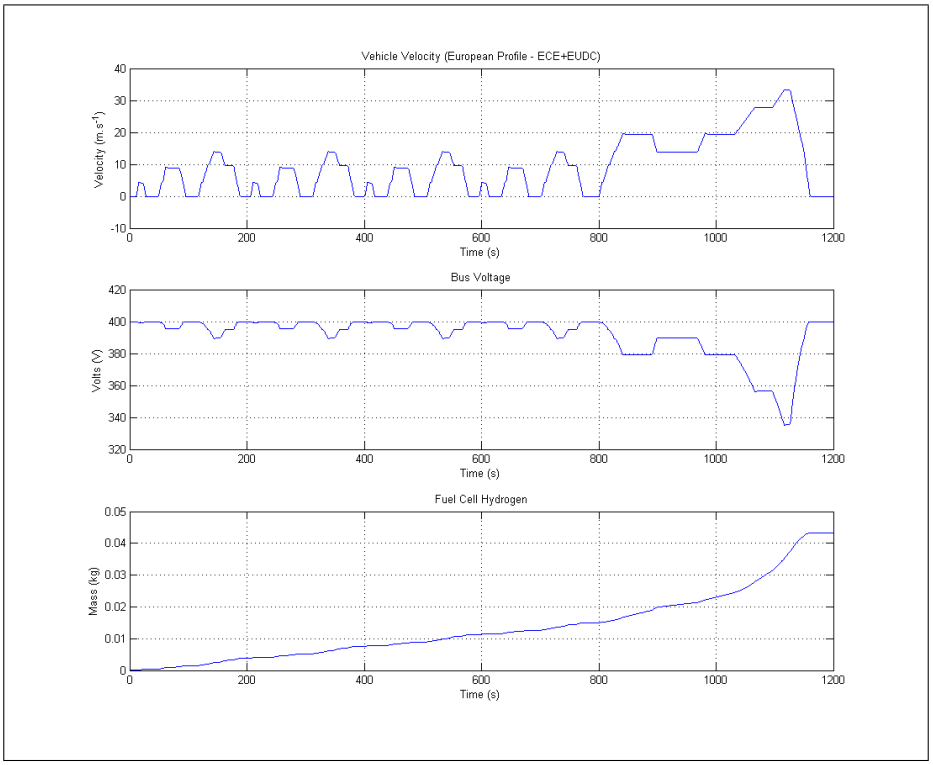



Figure 5.17: Drive cycle test results for energy management model showing the vehicle speed, bus voltage and hydrogen used by fuel cell.

A comparison of the hydrogen used by the fuel cell over these three legislative driving cycles is presented in Table 5.2. The lack of an energy management controller means that higher demands are placed on the fuel cell. In particular since the bus voltage is set by the vehicle speed and there is no regenerative braking implemented, whenever the vehicle slows down the ultracapacitor must be recharged by the fuel cell. This

explains the higher amount of hydrogen usage compared with the controlled version as shall be seen in the next section.

Table 5.2: Hydrogen usage and estimated range for different driving cycles in Dymola. No regeneration or energy management control. 2.1 kg stored Hydrogen.

| **Driving Cycle** | **Distance [km]** | **Hydrogen Used [g]** | **Consumption [g/km]** | **Range [km]** |
|---|---|---|---|---|
| NEDC | 11 | 43 | 3.9 | 538 |
| UDDS | 12 | 52 | 4.3 | 488 |
| US06 | 12.7 | 90 | 7.1 | 296 |

Given that the total available mass of hydrogen stored on-board the LifeCar vehicle is 2.1 kg, the vehicle range values can be calculated from the consumption ratio. In terms of stored hydrogen gas, 2.1 kg is equivalent to a tank size of 70 litres at a pressure of 35MPa, assuming a hydrogen gas density of 0.089g/l at ambient pressure. Since the UDDS driving cycle is more aggressive in that there are a larger amount of acceleration and deceleration events, this implies more charging and discharging of the ultracapacitor leading to increased current demands on the fuel cell. The US06 drive cycle has much steeper acceleration gradients with the a maximum acceleration of 3.8 $\mathrm{m.s^{-2}}$ which leads to increased torque demands from the electrical machines and associated increased power demand which is transferred to the fuel cell.

### Simulation of Translated Model in Simulink Environment

The ability to use the LifeCar plant model in the Simulink environment is tested in order to establish the feasibility of using this type of object-oriented model for controller design, which will invariably make use of the industry standard Simulink environment. In order to be able to export the Dymola energy management model into Simulink, the first step is to define the causality required by the controller within Simulink. This is done by defining the blue input connector arrows and the white output connector arrows as shown in Figure 5.18. When the model is translated by the Modelica translator in Dymola, the equations are sorted and manipulated in order to solve them for the specified outputs. In this way the model can then be compiled with the correct mapping of inputs and outputs. Dymola then generates a C-code file describing the model and a Matlab MEX file in order to allow Simulink

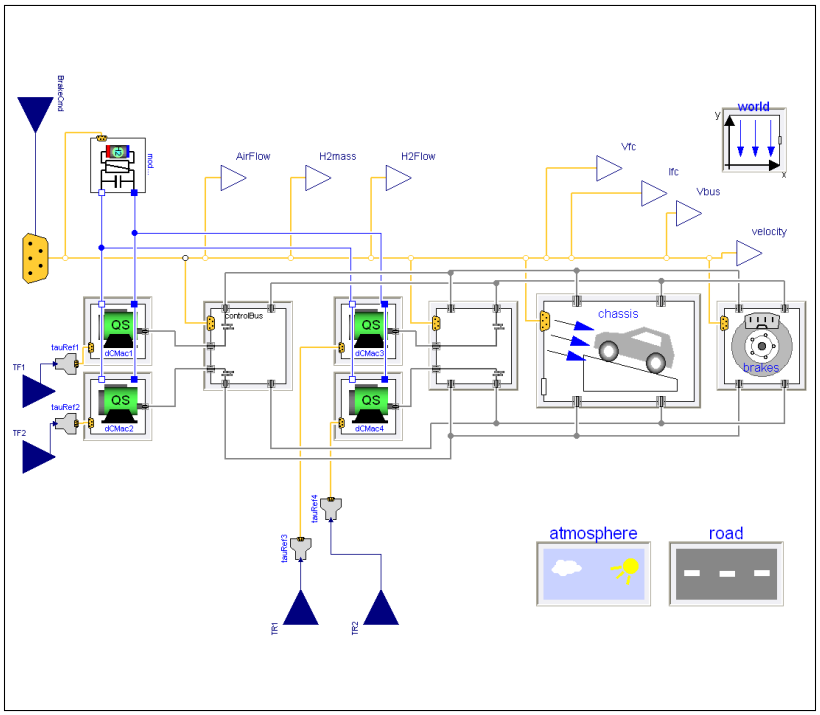



Figure 5.18: LifeCar energy management model prepared for export to Simulink.

to interface with the generated C-file and incorporate the system as a causal block in a standard Simulink model.

Figure 5.19 shows the LifeCar plant incorporated into a forward facing model with an energy management controller. The main functions implemented by the energy management controller in Simulink is to provide a regenerative braking strategy and power limit the vehicle when the ultracapacitors are empty so as to prevent excessive demands from the fuel cell. The regenerative braking strategy determines the amount of electrical machine braking as a function of the vehicle speed and ultracapacitor SOC, where maximum regeneration occurs when the SOC is below 80% and the vehicle speed is above 28km/h. The reason for this is that there is less potential to store energy when vehicle speeds are low or the ultracapacitor is nearly fully charge. This has an overall effect of increasing the usage of the ultracapacitor and thereby reducing the power demand on the fuel cell.

The benefit of the Simulink controller on the Dymola plant model's performance can be seen by comparing the simulated hydrogen usage results from the uncontrolled model in Table 5.2 with the controlled model in Table 5.3 column 2. For example, with the NEDC driving cycle, the Dymola simulation resulted in 43g of hydrogen usage while with the controller this was reduced to 29.4g. Hydrogen usage is reduced

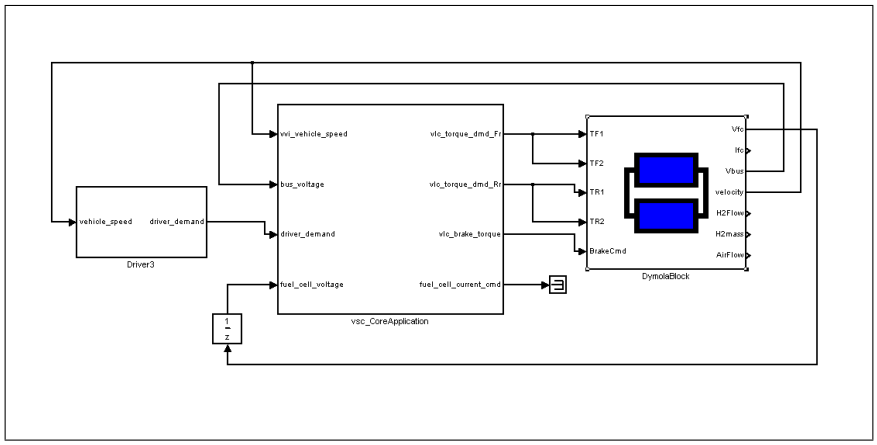



Figure 5.19: Dymola LifeCar plant model integration with Simulink controller.

by approximately 30% for each of the different driving cycle tests. This can be directly attributed to the decrease in fuel cell current in all drive cycles since the ultracapacitor recharching is not left to the fuel cell as it was in the model with no regenerative braking.

As a final means of validation and verification, the Dymola plant model was simulated in parallel with an existing validated Simulink energy management model for LifeCar. The Simulink controller was validated through tests performed on the prototype vehicle using a powertrain dynamometer to emulate a series of driving conditions as described in [154]. Comparison with the Simulink model provides a means of verifying that the results provided by the translated Dymola plant are in fact representative. Differences between the hydrogen usage calculated for the Dymola and Simulink plant models for the three legislative drive cycles tested are presented in Table 5.3.

Table 5.3: Hydrogen usage and estimated range for different driving cycle tests in Simulink. Comparison of Dymola and Simulink plant models with regeneration and energy management control.

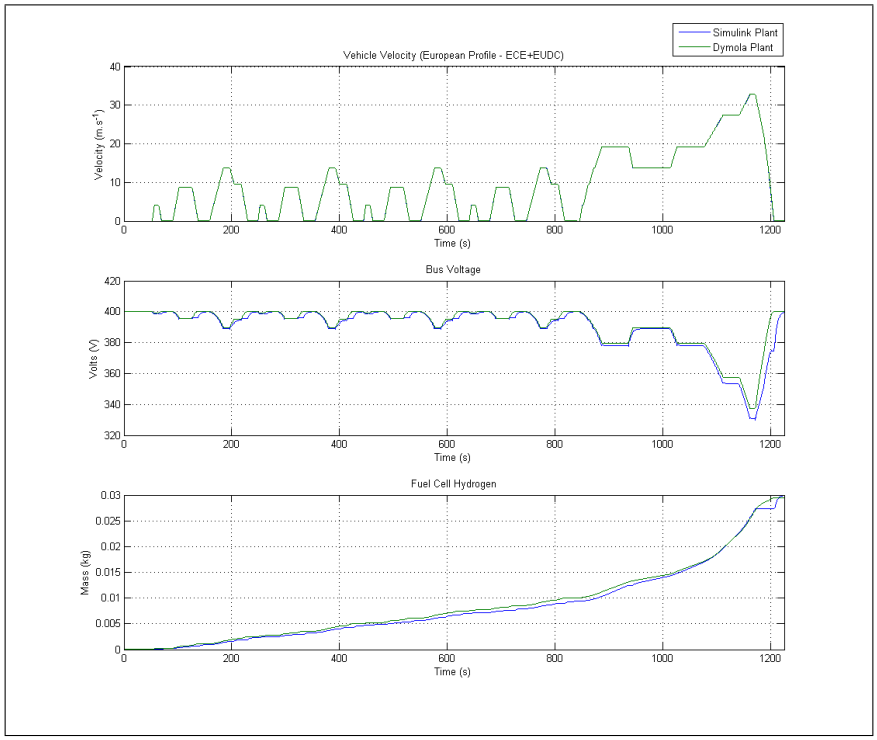
| **Driving** | **Hydrogen Used [g]** | | **Variation** | **Range [km]** | |
|---|---|---|---|---|---|
| **Cycle** | Simulink | Dymola | | Simulink | Dymola |
| NEDC | 29.9 | 29.4 | 1.6% | 766 | 780 |
| UDDS | 31 | 35 | 12.9% | 805 | 714 |
| US06 | 56 | 64 | 14.3% | 480 | 420 |

From Table 5.3 it can be seen that there is significant agreement between the two models for the NEDC drive cycle, with a variation of only 1.6%. In the case of the UDDS and US06 drive cycle tests, the variation in hydorgen usage is seen to increase by over 10%. The main reason for this difference is that the Simulink energy management controller has a control loop around the bus voltage which provides a current command to the converter controller. This allows for more precise control of the fuel cell current and consequently more efficient usage of the ultracapacitor as a power supply. In the Dymola plant, the converter model does not accept a current control command but is implemented in such a way that the load voltage is enforced according to the given reference, and the current at the supply side is calculated in order to balance the energy of the input with that of the output. In other words, with the Dymola plant, the ultracapacitor voltage is set by the vehicle speed and the fuel cell power must equal the ultracapacitor power.

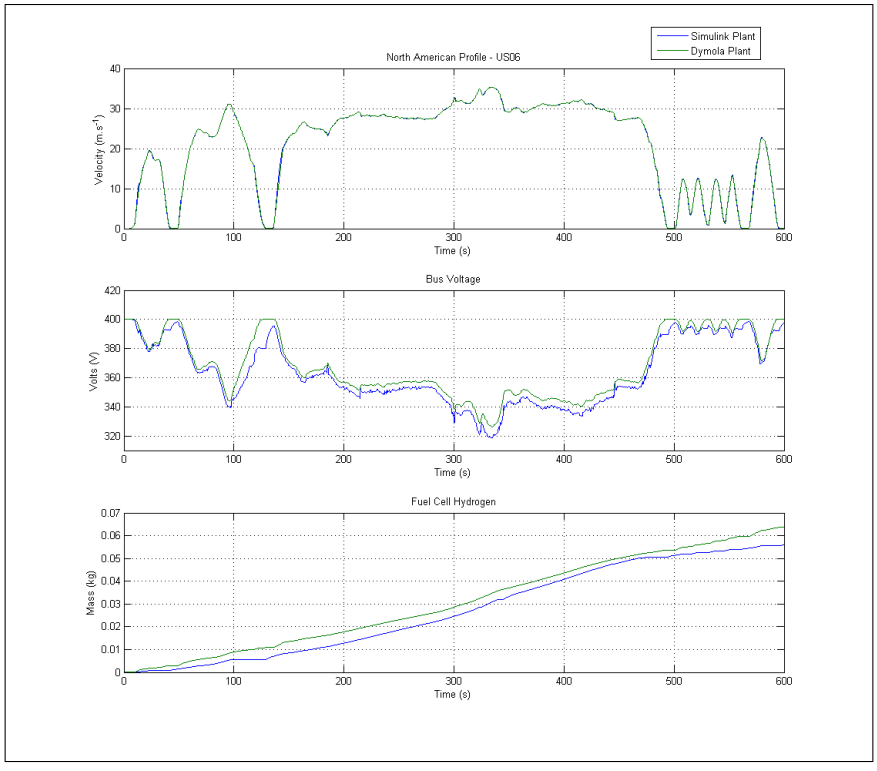
Figure 5.20(a) shows a comparison of the vehicle speed, bus voltage and hydrogen mass, for the NEDC drive cycle simulation results. This plot shows that the bus voltage is on average kept high during this drive cycle due to the relatively low vehicle speed and therefore there is not much usage of the ultracapacitor as a power supply. Since the bulk of the power is supplied by the fuel cell, the functionality of both plant models is similar.

Figure 5.20(b) shows the same comparison for the US06 drive cycle. Here it can be seen that the Dymola model is not able to make as efficient use of the ultracapacitors since the bus voltage is brought back to 400 V sooner than in the controlled Simulink model. Ultimately, this means that when the drive cycle contains more acceleration and deceleration events, the potential for energy recapture is reduced since the ultracapacitor voltage is brought back to the upper limit much sooner than necessary. Further in order to maintain the higher average voltage on the bus, the fuel cell current must increase since its voltage cannot and hence the average hydrogen usage increases.

In order to correct this error, a new version of the converter model must be implemented that enables better control of its functionality. An example of this is shown in Chapter 6 where a converter with a controlled power split is developed.




(a) NEDC drive cycle




(b) US06 drive cycle

Figure 5.20: Comparison of Dymola and Simulink plant model results for an NEDC and US06 drive cycles.

### 5.3.3 High Fidelity Model Specialization

In order to test the flexibility of the modelling method to make changes to the developed HEV library and LifeCar partial and base models, it was decided to develop a second powertrain model with some higher fidelity subsystems that could be used for performance testing. The main purpose being to test how much of the previously developed models could be reused and the effort required to implement fidelity changes within the vehicle subsystem models. In this case the intention is to examine the effects of weight transfer, tyre slip and driveline shaft flexibility on the vehicle's longitudinal dynamics under tip-in and tip-out conditions. The modelling steps of the *Implementation* stage of the method are repeated in order to specialize the subsystem base models to a level suitable for the performance tests. In particular, high fidelity packages were added within the driveline, and electrical machine base subsystem packages, while the chassis model was replaced with a higher fidelity version from the available libraries. Also a new wheel subsystem model was created in order to model non-linear tyre characteristics.

An overall comparison of the energy management model and the high fidelity model is given by the pole-zero map in Figure 5.21 and Table 5.4. Figure 5.21 shows that the frequency range of the plant model is increased by a factor of 10 from 21 $Hz$ to 212 $Hz$. In Table 5.4 it can be seen that the C-code generated when compiling the high fidelity plant model is 40% larger than that of the energy management model. Also comparing the runtime performance of a test simulation using a 200s ECE drive cycle shows that simulating the high fidelity model requires almost 5 times more CPU time than the energy management model.

Table 5.4: Comparison of High Fidelity and Energy management plant model performance

| Plant Model | Lines of C-code | CPU time for ECE cycle |
|---|---|---|
| Energy Management | 8453 | 2.27 s |
| High Fidelity | 12865 | 10.66 s |

An explanation of the changes made to these subsystem models is presented in the following subsections.

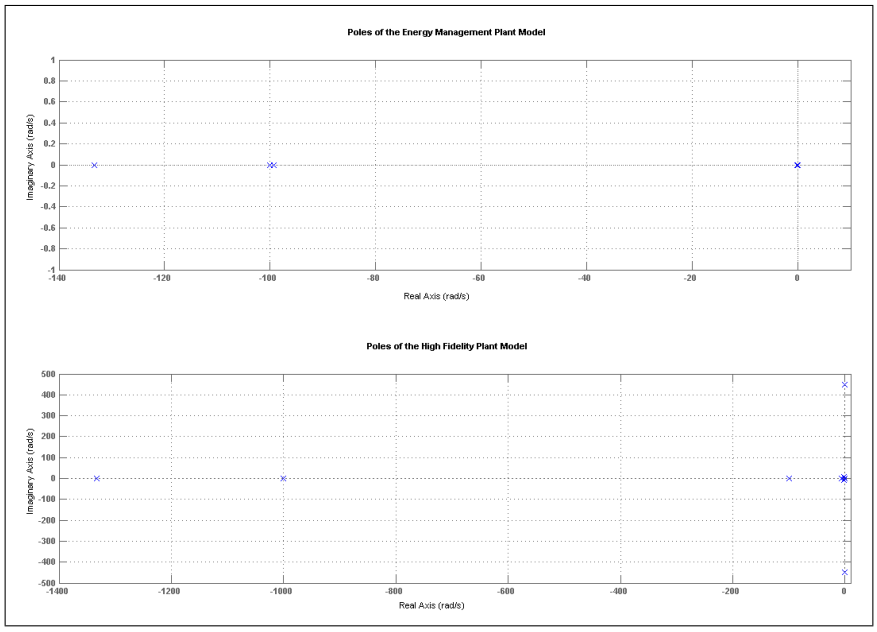



Figure 5.21: Pole-zero maps comparing the Energy management and High Fidelity plant models linearised under initial conditions at time $t = 0$.

### Driveline

Once again the base driveline models were specialized, only this time the ideal gears in the base model are replaced with flexible shafts as shown in Figure 5.22. The flexible shaft model is essentially two inertias connected by a rotational spring and damper system. A parameter input for the shaft inertia is divided equally between the two inertias while the spring-damper system provides the stiffness and damping constants necessary to examine the effects of shaft flexibility on vehicle acceleration. By applying the spring-damper system, the relation of the torque ($\tau$) and relative rotation angle ($\phi_{rel}$) between the two ends ($a$ and $b$) is modified according to equations (5.16)–(5.18).

$$\phi_{rel} = \phi_b - \phi_a \tag{5.16}$$

$$\omega_{rel} = \dot{\phi}_{rel} \tag{5.17}$$

$$\tau = k\,\phi_{rel} + d\,\omega_{rel} \tag{5.18}$$

where: $k$ is the stiffness constant, $d$ is the damping constant and $\omega_{rel}$ is the relative angular velocity.

On the LifeCar vehicle, the actual drive shafts where very short as the driving motors where mounted as close as possible to each wheel. This meant that the inertia of

the shaft was very small compared with the inertias of the motor shaft and wheel that it connected, also the damping characteristics of this shaft where not known. For the purposes of this study no damping constant was used while stiffness and inertia values were calculated from the physical dimensions of the shaft.

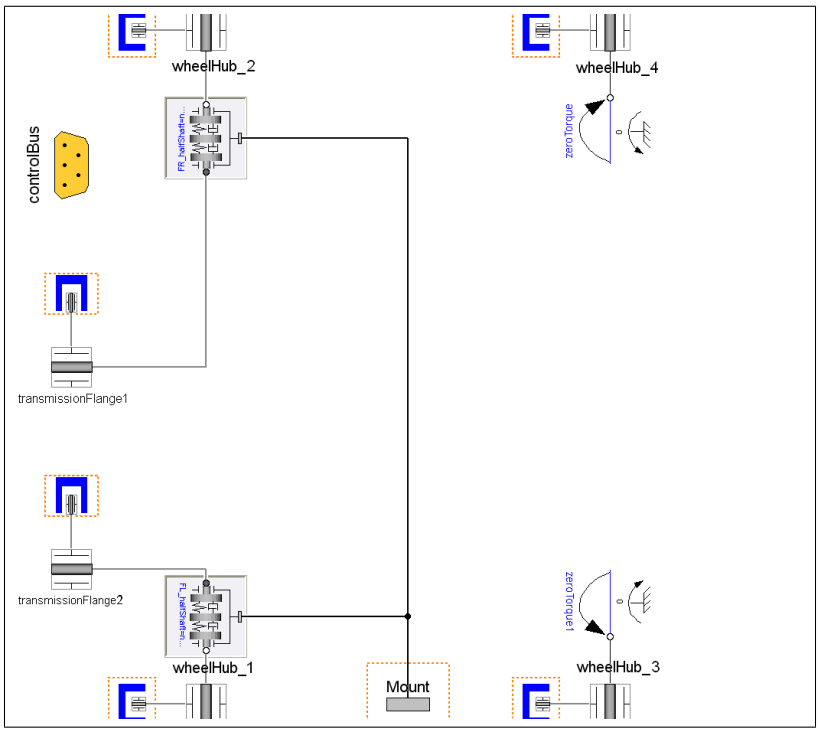



Figure 5.22: High fidelity driveline model with flexible shafts.

Additionally, for forward compatibility, the shaft components were attached to the driveline mount so that future applications could make use of this subsystem model within a future multibody dynamics model. This inclusion adds no overhead in terms of code or calculations to the final simulation model since it is included as an optional connection as shown by the dashed box around the mount block in Figure 5.22. In other words, the model developer needs to specify the inclusion of multibody dynamics by setting a boolean parameter in the top level of the model to logical "true". If this is not done, then all multibody connections, components and associated code are ignored when the final model is compiled for simulation.

### Electrical Machines

Since the high fidelity performance tests will also look at transient mechanical effects in the driveline, it was decided to replace the base electrical machine models with

controlled transient DC machines from the SED library. Unlike the quasi-static variants used for the energy management model, the transient model allows the user more access to the control variables and is designed for parameter calibration. Using this higher fidelity model in the performance testing provides a means of calibrating the model performance to known results so that they match the real machines more closely.

### Chassis and Tyre Models

The final extension to the LifeCar powertrain model for the high fidelity test was to replace the chassis model with another version from the PowerTrain library containing a half-car suspension model. This new chassis model requires suspension stiffness ($k_{f,r}$) and damping ($c_{f,r}$) rates, vehicle mass distribution between front ($m_f$) and rear ($m_r$) axles, the length of the wheelbase ($a + b$) and the height ($h$) of the centre of gravity (CoG) as seen in Figure 5.23. In order to parameterize this model, measured values from the prototype vehicle were used where available. Specifically the vehicle mass distribution was taken from the validated Simulink model, while the suspension data was not available and therefore the default values supplied by the PowerTrain library were used. The suspension model is used to determine the weight transfer effects of the vehicle due to changes in the pitch angle ($\theta$) while accelerating and braking. This is needed to determine the load force ($\mathrm{F_Z}$) on front and rear wheels which in turn is required by the tyre-slip model to determine the friction for each wheel and hence the driving force ($\mathrm{F_X}$). Further details on the mathematical derivations in a half-car suspension model can be found in [155, 156].

Within the chassis model it is necessary to instantiate an environmental loss model and four tyre models. For the purposes of this study, the same drag model as for the energy management powertrain model was used, which includes aerodynamic drag and rolling resistance. A new tyre-slip model was developed to assess the wheel slip during a tip-in and tip-out performance simulation. The powertrain library made available a linear tyre-slip model that proportioned the driving force to the slip ratio by a stiffness constant, and two non-linear tyre models. The first non-linear model is based on the model proposed by Rill [157, 158], which requires steady state force and slip parameters under upper and lower weight conditions, and the second is based on the commonly cited Pacejka magic formula [159, 160]. In order to evaluate the tyre-slip on different road surfaces, such as dry or wet, the new non-linear tyre

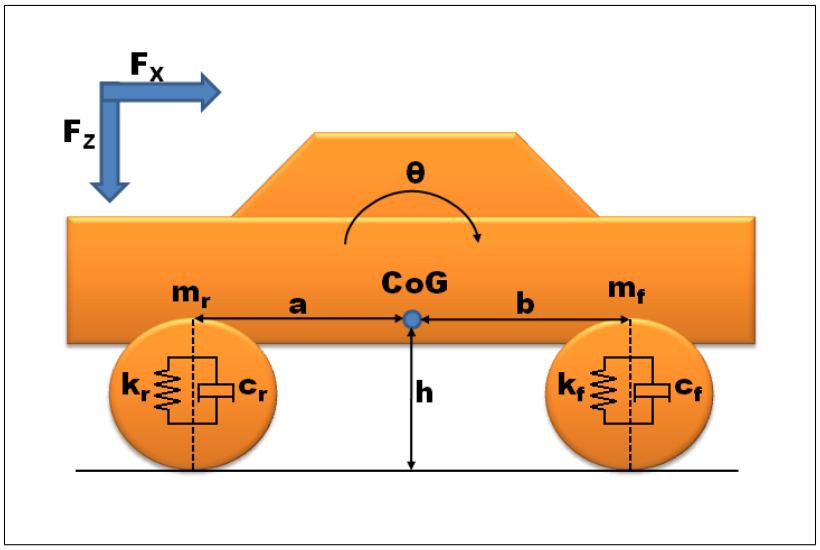



Figure 5.23: Free body diagram for the half-car suspension model.

model was based on Pacejka model but modified in order to allow parameterization according to the coefficients in (5.20). Specifically the tyre model calculates the tyre slip ratio ($\sigma$) and the friction coefficient ($\mu$) according to equations (5.19) and (5.20) respectively.

$$\sigma_{(f,r)} = \frac{(\omega r_w)_{(f,r)} - v}{\text{MAX}\big(v, (\omega r_w)\big)} \tag{5.19}$$

$$\mu(\sigma)_{(f,r)} = \text{D}\ \sin\left(\text{C}\ \arctan\left(\text{B}\sigma_{(f,r)} - \text{E}\Big(\text{B}\sigma_{(f,r)} - \arctan\left(\text{B}\sigma_{(f,r)}\right)\Big)\right)\right) \tag{5.20}$$

where: the subscript $_{(f,r)}$ represents front and rear, $v$ is the vehicle velocity and $(\omega r_w)$ is the rolling velocity of the wheel given by the product of angular wheel velocity and wheel radius. The MAX function is used to calculate the slip ratio under both acceleration and braking conditions [161].

The value of slip ratio can be interpreted in the following way:

$\sigma > 0$ indicates the wheel velocity is greater than the vehicle velocity $\Rightarrow$ accelerating.

$\sigma = 1$ indicates a much higher wheel velocity than vehicle velocity $\Rightarrow$ "spinning".

$\sigma < 0$ indicates the vehicle velocity is greater than the wheel velocity $\Rightarrow$ braking.

$\sigma = -1$ indicates a much higher vehicle velocity than wheel velocity $\Rightarrow$ "locked".

With the load force from the chassis model and the friction from the tyre model, the driving force ($\text{F}_\text{X}$) can then be calculated according to equation (5.21).

$$\text{F}_{\text{X}(f,r)} = \mu_{(f,r)}\ \text{F}_{\text{Z}(f,r)} \tag{5.21}$$

The coefficients B, C, D and E in equation (5.20) can be considered form factors which describe the shape, peak, slope and curvature of the graph relating friction

to slip. For the purpose of this study three parameter sets for different road conditions [161, 162] were used. A plot showing the friction-slip relationship for the "Dry Tarmac", "Gravel" and "Ice" road surfaces is the shown in Figure 5.24. Further details on the modelling of tyre and vehicle dynamics can be found in [156, 159, 163].

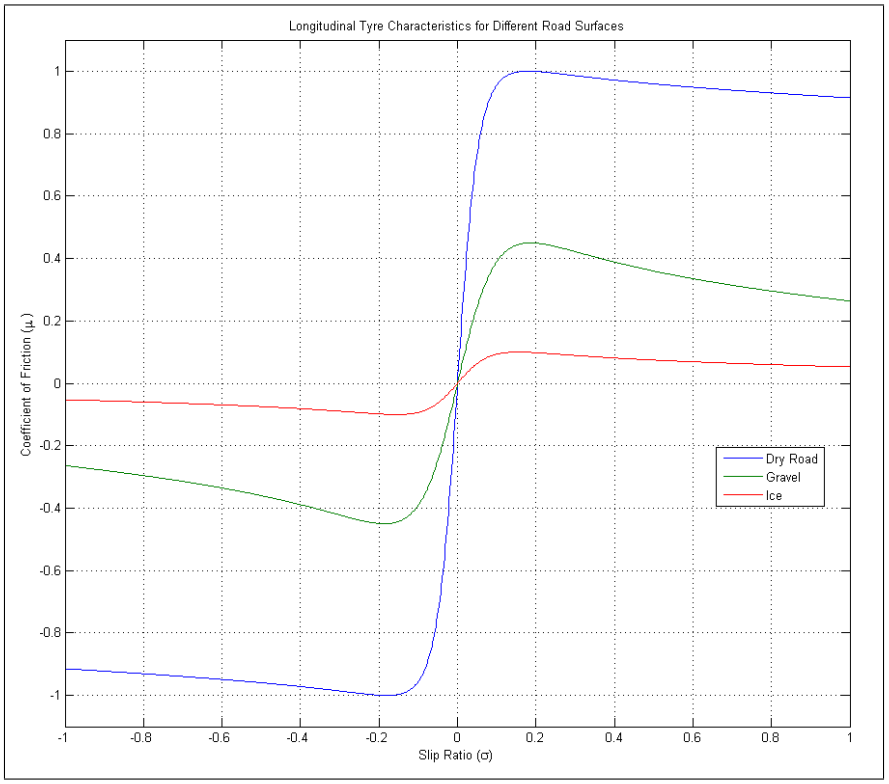



Figure 5.24: Friction versus slip for different road conditions.

### Powertrain

In order to create the final executable model, the base vehicle model shown in Figure 5.11 is inherited and the subsystems are replaced with the high fidelity subsystems to produce the model shown in Figure 5.25. For these tests no drive cycle or driver model was included, instead accelerator and brake commands are converted directly to a torque input reference for the electrical machines and a separate friction brake input is also provided. This is done to mimic the action of the energy management controller which would divide the brake torque demand between regenerative braking and friction braking. As mentioned in Section 5.3.2, the braking split to the electrical machines is dependant on the SOC of the ultracapacitor and

the vehicle speed, with the balance of the demanded braking torque being supplied by the friction brakes.

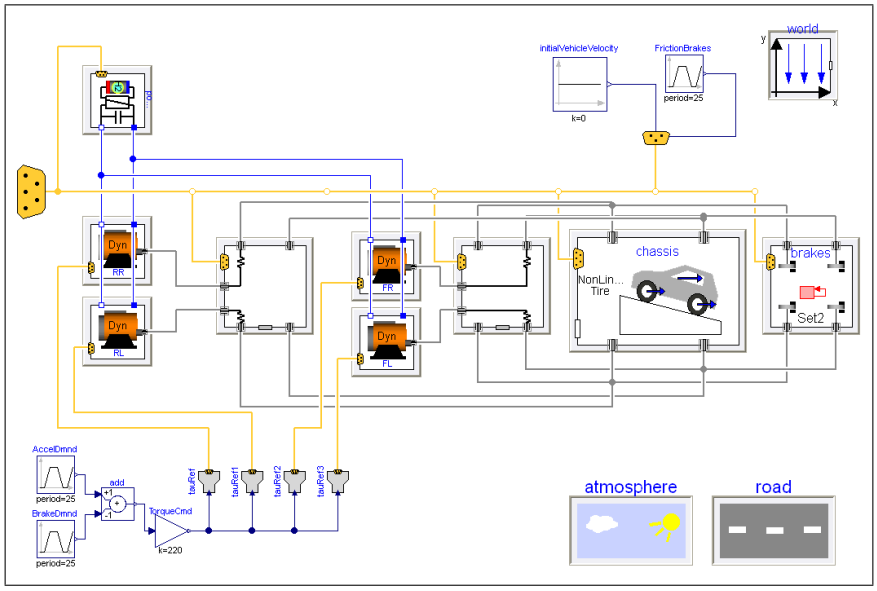



Figure 5.25: Executable high fidelity model of LifeCar powertrain for Tip-in Tip-out testing.

### 5.3.4 High Fidelity Model Validation Tests

For this particular model, the main aim was to determine the effect that the high fidelity subsystem models had on the longitudinal dynamics of the vehicle. To do this, an open-loop test is performed on the high fidelity powertrain model within the Dymola environment using different tyre model parameterizations to simulate different road surfaces.

#### Simulation Tests in Dymola Environment

Three sets of simulations are performed with the tyre model parameters corresponding to the road surfaces shown in Figure 5.24. For all tests, the chassis model is parameterized with a mass distribution of 388.8 kg at the rear and 305.4 kg at the front of the vehicle, a wheelbase length of 2.45 m and a CoG height of 0.4 m. During the gravel and ice road surface tests, the input torque demand was scaled down from the maximum 220 N.m until the point where wheel "spinning" first begins.

### Dry Road Surface

A tip-in tip-out demand is applied to the vehicle by providing a step input in torque to each of the four electrical machines at their maximum rated value of 220 N.m and then demanding a braking torque of 250 N.m with the difference supplied by the friction brakes. The vehicle speed and torque demands (per wheel) are shown in the first two plots of Figure 5.26. Here it can bee seen that the main braking torque is initially provided by the electrical machines until regeneration is no longer feasible. Following this the friction brakes supply all the demanded torque. Regenerative braking starts decreasing when the vehicle speed reaches 10m.s$^{-1}$ and stops at speeds below 5 m.s$^{-1}$. This is because the kinetic energy of the vehicle is less than 2.5 Wh at 5 m.s$^{-1}$ and the generation efficiency of the electrical machines is lowest at low speeds [146, 164]. For a given torque the electrical machines power is proportional to the speed but the electrical and mechanical losses are relatively constant, therefore at low speeds the losses are comparable to the output power and therefore reduce the efficiency [165]. While regeneration decreases, the friction braking increases to meet the braking torque demand.

Examining the net torque demand shown in the middle plot of Figure 5.26, it can be seen that there is a torque disturbance when the regenerative braking decreases and the friction braking increases. This is due to the motor torque decreasing exponentially while the friction brake torque increases linearly. Further, torque oscillations are visible whenever a torque transient occurs. The lower plot in Figure 5.26 shows a closer view of the torque oscillations at the wheel shaft between 13.75 s and 13.8 s. From this it can be seen that there are three oscillation periods between 13.754 s and 13.795 s which gives a frequency of 73.17 $Hz$.

This frequency can be described as the resonant frequency of a dual inertia system [166], where two inertias are coupled by a flexible shaft. The resonant frequency ($\omega_n$) can be described in terms of the shaft torsional stiffness ($k$) and the inertias on either end of the shaft as shown in equation (5.22).

$$\omega_n = \sqrt{k\left(\frac{1}{J_r} + \frac{1}{J_w}\right)} \tag{5.22}$$

where: $J_r$ is the electrical machine's rotor inertia and $J_w$ is the wheel inertia.

Applying the model parameters to equation (5.22), where the motor shaft inertia and wheel inertia are 0.04 kg.m$^2$ and 0.8 kg.m$^2$ respectively, and the shaft stiffness is 8023 N.m/rad, gives a resonant frequency of 73 $Hz$.

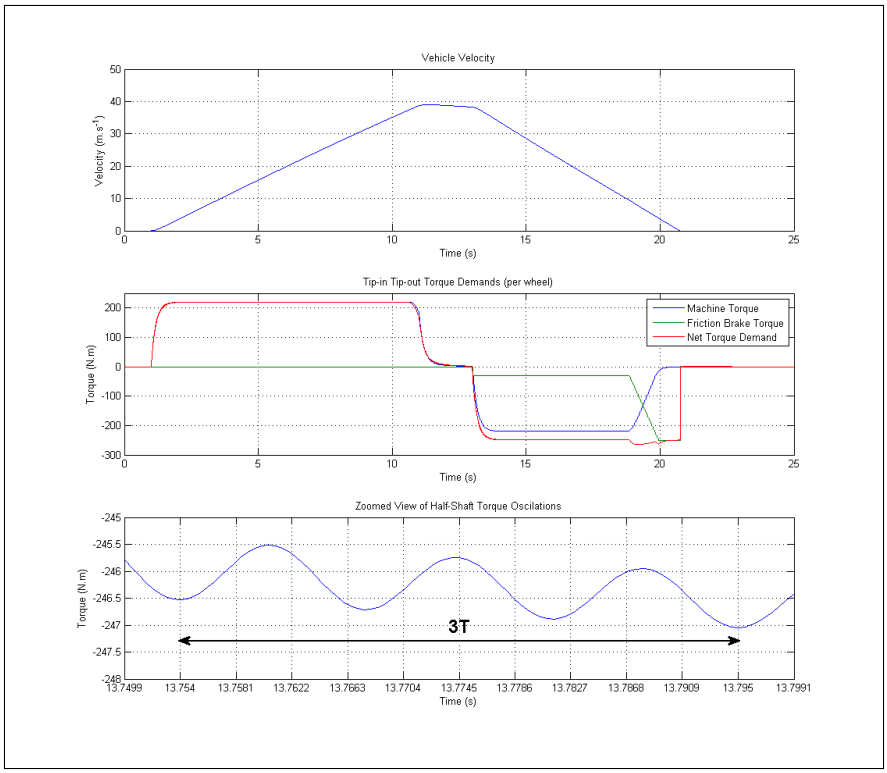
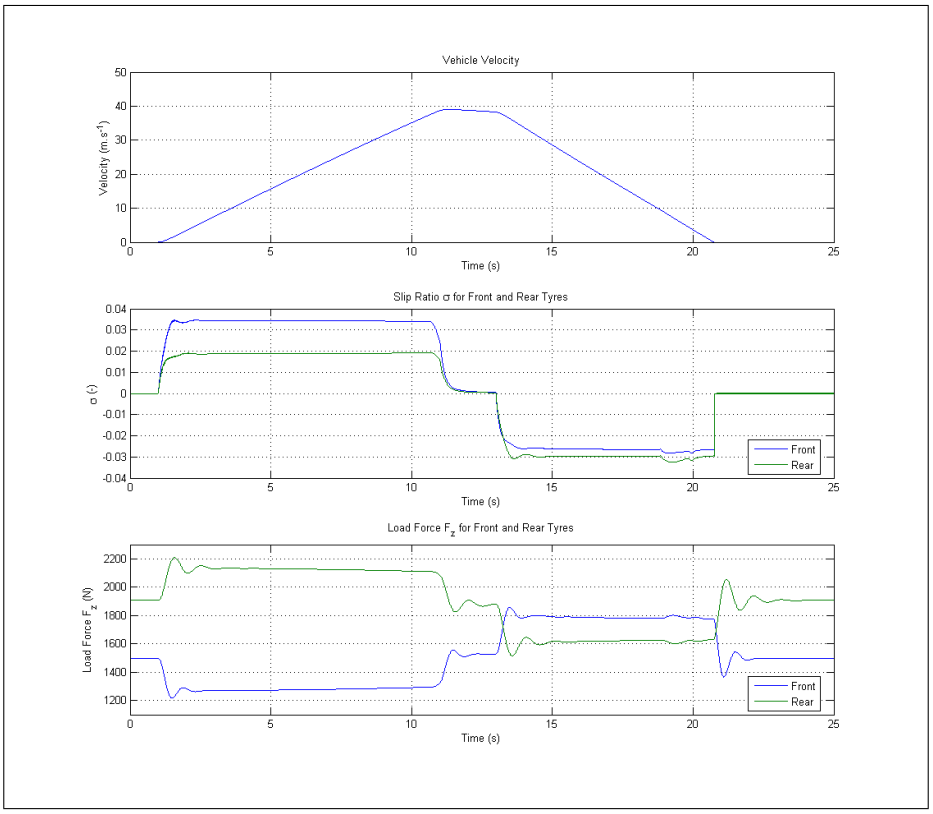



Figure 5.26: High fidelity torque demand and oscillation due to flexible half-shafts.

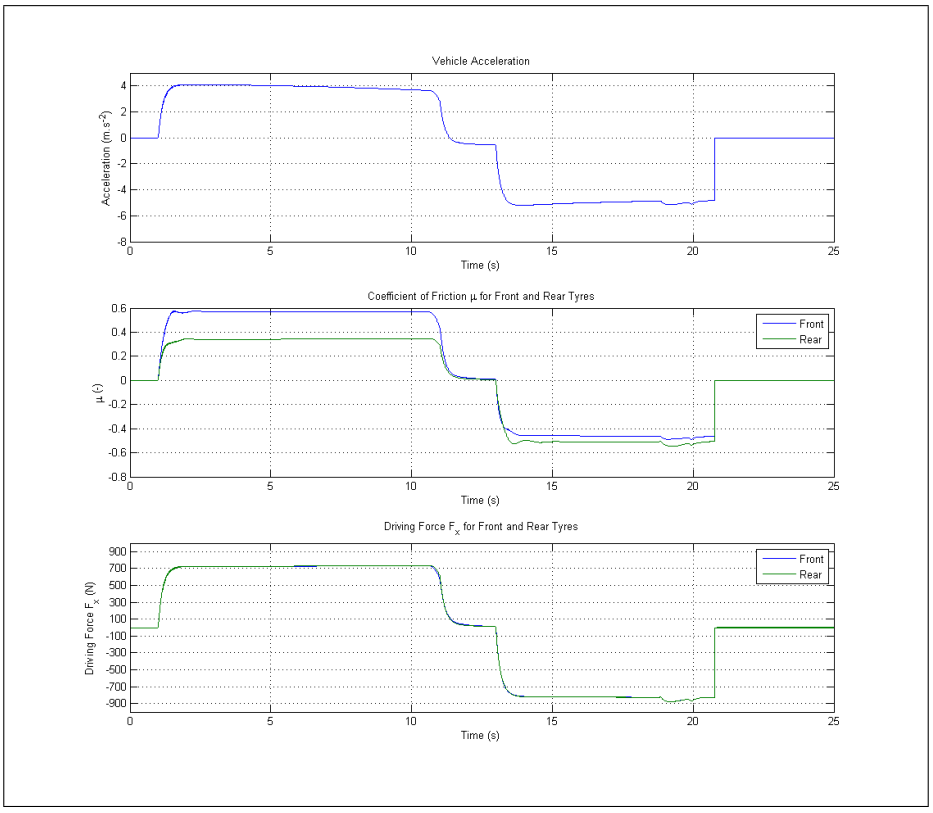
The effect of the high fidelity chassis and tyre models on the vehicle's longitudinal dynamics can be seen in Figure 5.27.

In Figure 5.27(a) it can be seen that during the 10 s tip-in period, the vehicle accelerates from $0 - 30\,\mathrm{m.s^{-1}}$ (100 km/h) in approximately 7 s and reaches a top speed of $39\,\mathrm{m.s^{-1}}$ (140 km/h). While accelerating, the vehicle is said to "squat" and the load force increases on the rear wheels and decreases on the front wheels due to the weight transfer in the chassis model. This effect is reflected in the tyre models by the slip ratio being higher for the wheels with less load on them. During braking the vehicle is said to "pitch" or "dive" as the weight is transferred to the front of the vehicle, thereby increasing the front load and decreasing the rear load. Also, the slip ratio is negative during braking and again the ratio is higher for the wheels with less load.

Since the vehicle must have a resultant force given by $\mathrm{F} = \mathrm{ma}$, the driving force at each wheel must be equal to $\mathrm{F_X} = m_v\, a/4$. Therefore, according to equation (5.21), the wheels with less load require a higher coefficient of friction. Figure 5.27(b) shows that the wheels with a higher slip ratio have a corresponding higher coefficient of friction in order to produce the same driving force.




(a) Tyre-slip and Load force




(b) Friction and Driving force

Figure 5.27: High fidelity Tip-In Tip-out simulation on dry road surface.

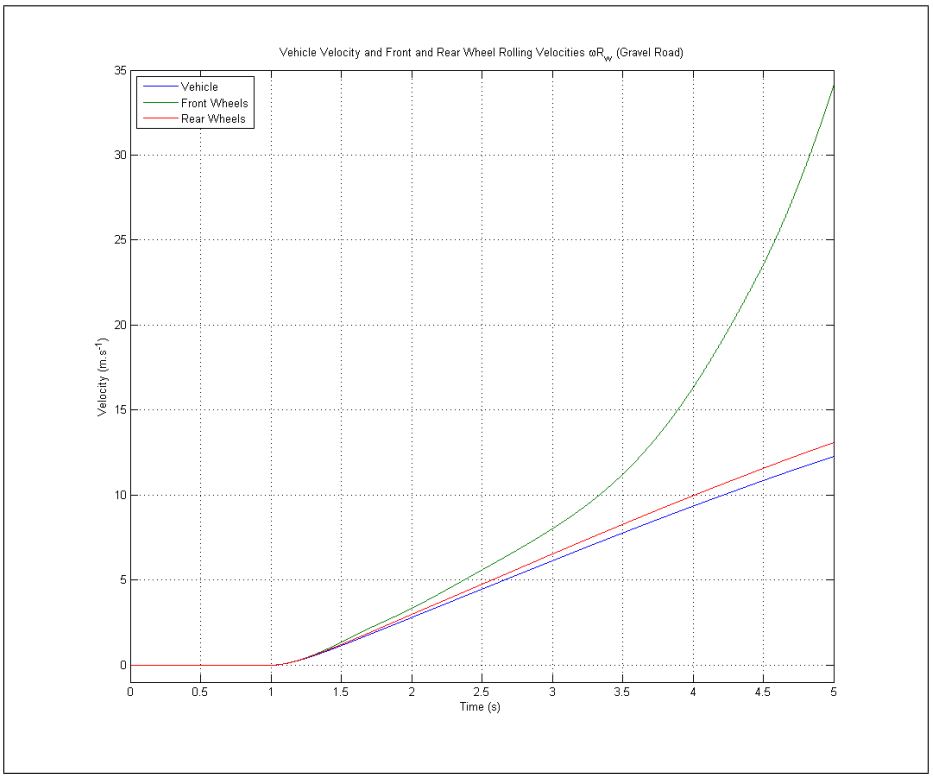
### Gravel Road Surface

For the gravel surface test it was found that for a torque step of 176 N.m, 80% of the maximum, the front wheels can no longer achieve the required friction and begin to spin as seen in Figure 5.28. Figure 5.28(a) shows that the front wheel rolling velocity increases exponentially in comparison with the vehicle velocity meaning a rapidly increasing slip ratio. In Figure 5.28(b) it can be seen that the friction coefficient of the front wheels reaches the peak value for gravel after 1.7 s ($\hat{\mu}_{gravel} = 0.45$), at which point any further increase in slip ratio leads to a decrease in friction and therefore a decrease in the driving force. After 5 s, the slip ratio of the front wheels reaches 100% ($\sigma = 1$) and the simulation terminates soon after since the electrical machines cannot support any further increase in speed.

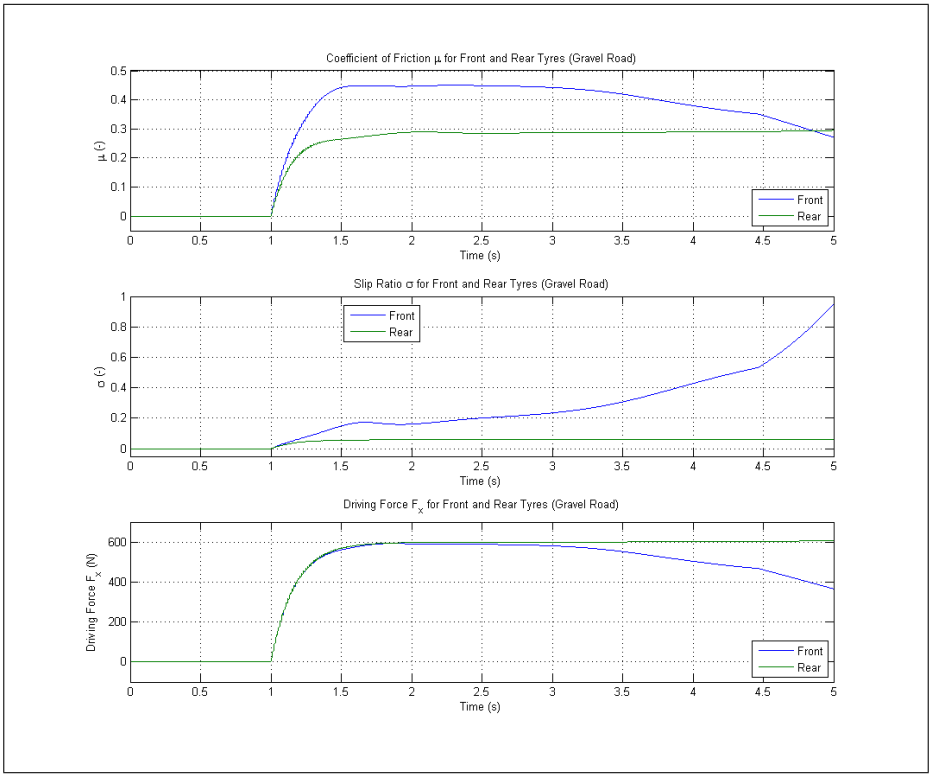
### Ice Road Surface

Similar results are found for the ice surface test only that in this case the maximum torque step input before wheel spinning occurs is at 44 N.m or 20% of maximum torque. Results are presented in Figure 5.29, where Figure 5.29(a) shows the vehicle and wheel velocities and also the torque at the wheels. Once again the torque oscillations can be observed only this time with the effects are not damped out by the spinning front tyres. With 11 periods between 4.15 s and 4.3 s, the frequency is still shown to be the resonant frequency of 73 $Hz$. Figure 5.29(b) shows that after 2 s the peak friction coefficient for the ice surface ($\hat{\mu}_{ice} = 0.1$) is reached by the front tyres. The friction then diminishes as the slip ratio increases and "grip" is lost causing the driving force on the front tyres to decrease.

In real world driving conditions, this corresponds to a driver accelerating hard from standstill and the wheels starting to spin. The driver would then ease off the accelerator pedal, reducing the slip ratio and returning to a higher value of friction, until traction is restored.

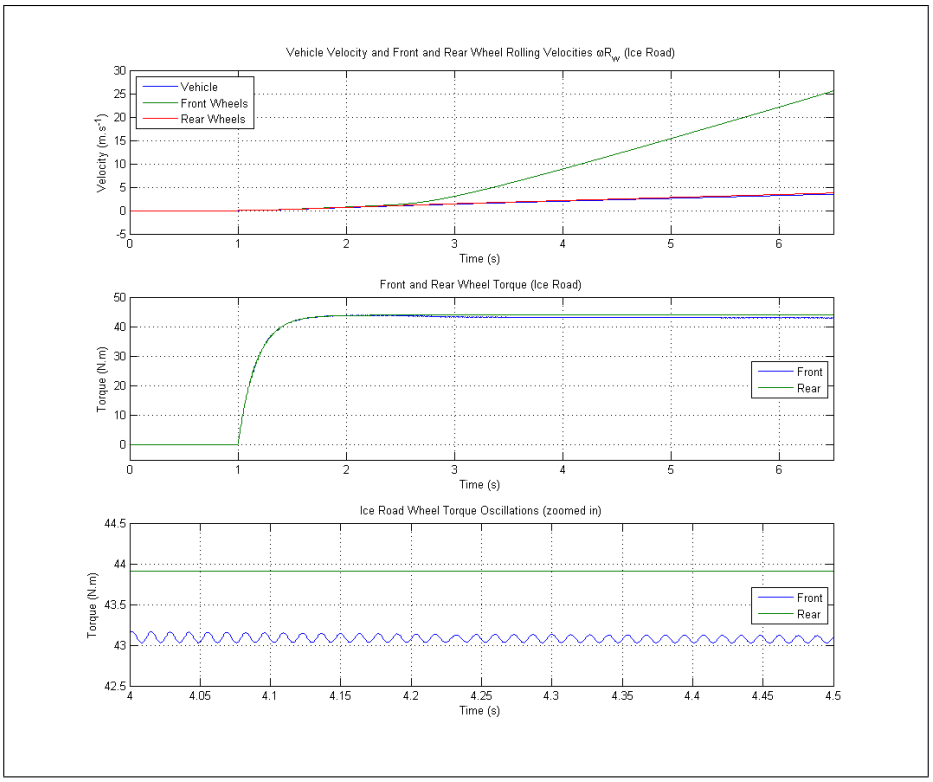



(a) Vehicle and Rolling Velocities

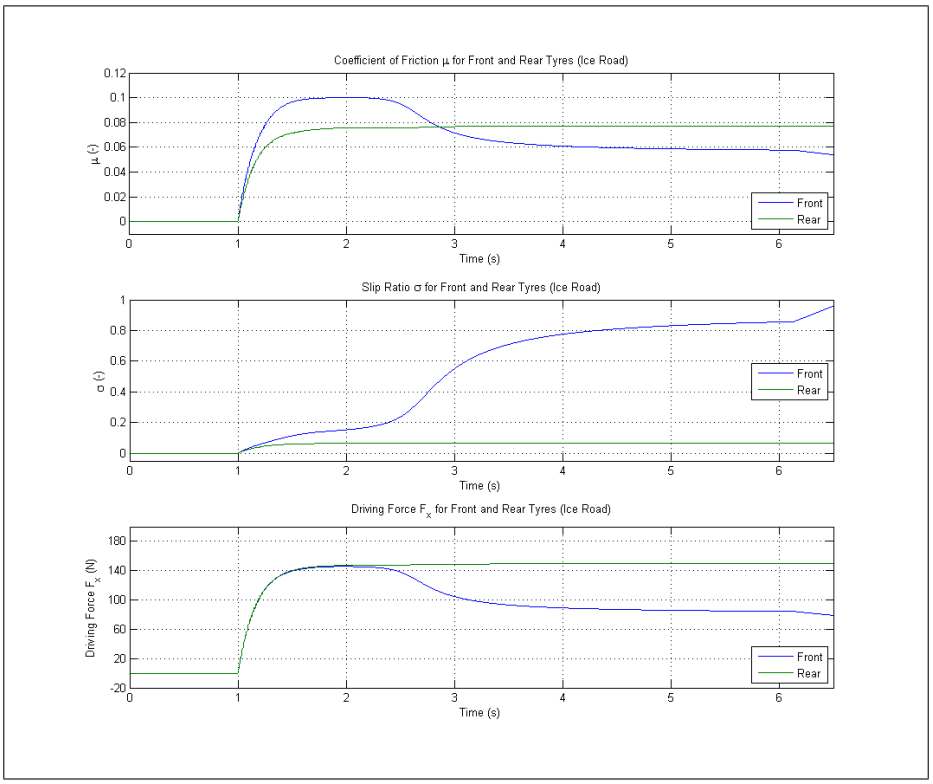



(b) $\mu$, $\sigma$ and $F_X$ for Front and Rear Tyres

Figure 5.28: High fidelity simulation results for gravel surface.




(a) Vehicle Velocity, Rolling Velocities and Wheel Torques




(b) $\mu$, $\sigma$ and $F_X$ for Front and Rear Tyres

Figure 5.29: High fidelity simulation results for ice surface.

## 5.4 Discussion

In all modelling domains it is necessary to decide what level of complexity or fidelity will be used when constructing a model.

Reviewing the construction of the model library during the modelling stage of this case study, design decisions and trade-offs in the following areas need careful consideration:

1. Partial Model Construction

   Deciding on a point or level where an item should be generic is not trivial. Though both the properties of reusability and extendability are desired, it stands to reason that reusability favours faster design and extending libraries should only be done when necessary. Having said this, it is also not possible or practical to have a library with an exhaustive set of components and subsystems to be reused. For a design where the technology and know-how is in an advanced stage, various components and subsystems will become standard and should form part of a generic library. Designs that are part of the initial stages of the development of a certain technology will more likely require a lower level of reusable subsystems and a higher degree of extendability.

2. "Off-The-Shelf" Model Libraries

   Independently validated models in specialized areas are developed by third parties, some freely available and others sold commercially. Though the availability of tested models is key to faster development of new technologies such as HEVs, there are also associated risks when incorporating such models within a development process. One of these was encountered during this case study in the form of encrypted models. Use of such models places certain limitations on the modelling process thereby reducing the flexibility of the overall design.

   Having said this, in some cases, where there are no open source alternatives or in-house expertise, it may be more beneficial from a time-cost perspective to purchase third party models rather than develop them. Equally, open source solutions are not without their shortcomings such as multiple version changes and poor documentation. This is why it is suggested by the author that a well managed central repository of base models should be available so that all developers can commence at a common point and incorporate the re-

quired fidelity. If the end users then choose to share their upgraded models by submitting them to the repository, the value of the common repository is increased for all users. Various compromises made when using off-the-shelf software are discussed further in Chapter 7-Model Management.

3. Model Abstraction

   When breaking down a system into its subsystems and their respective functions, or a program into classes and their underlying methods, one of the decisions to be made is how much to include in a particular subsystem model. In particular, the inclusion of controller functionality should be done in a modular manner, keeping the physical model separate from the control logic as far as possible. Developing a model where the physical plant is totally separate from controller models leads to a more flexible design since the plant model can be used independently and the control logic can be modified without affecting the plant models functionality. For example, in Section 5.2.4 the base model for LifeCar was developed without incorporating a driver model allowing for the same base to be used for various simulation models requiring different driver types as well as exporting to external software such as Simulink. This point is of particular importance when an acausal modelling language is being used since the plant model can be described by physical equations which have no causality but a controller performs specific functions on its inputs to give a required output and is thus causal by nature. Incorporating plant and controller into one integrated model therefore reduces the benefits of acausal modelling since the causality of the controller model will dictate a causality for the overall model.

Pidd [167] proposes a simulation modelling doctrine composed of five principles to help overcome some of the challenges associated with the development process.

1. The first principle suggests that it is not necessary to build a complex model to describe a complex system, if the user's knowledge and expertise is included as part of the "requisite variety" of the modelling process. Meaning that the modelling requirements are not only met by the model itself, but by the model, its assumptions and the critical analysis of its results.

2. The second principle stresses the importance of the incremental development of models. This allows for all modelling to initiate with a simple model from

which some lesson is learnt in order to then refine the model. This also satisfies the requirements of "Occam's razor" since a developer can stop when the model provides the required output without spending time and effort on more complex modelling where it is not needed.

3. The third principle is modularize model development in order to avoid creating large and complex models which are both difficult to use and understand. This is the same philosophy behind object-oriented software development.

4. The fourth principle deals with the use of data in model development and stresses that modelling should not be driven by the availability of data, but instead data should be sourced as needed for the model.

5. The fifth and final principle is an observation that modelling activities do not progress linearly even if the first four principles are followed. This is due to the way in which developers need to continuously divide their time between understanding the problem context and producing models throughout the incremental modelling process, often having parallel lines of thought.

Object-oriented principles were devised to help cope with the growing complexity of programs and are therefore well suited to dealing with the analysis and construction of complex systems. Abstraction and inheritance link well with the modelling principles suggested by Pidd [167], in that they promote problem simplification and the ability to add complexity where it is needed.

## 5.5 Conclusion

Seeing that a HEV is a product that combines systems from multiple engineering disciplines, it is appropriate to implement a modelling method that promotes component orientated modelling of complex multi-domain systems. This case study demonstrated that the proposed modelling method contributes to HEV development in the following ways:

- It provides a structured multi-domain development approach that allows developers to focus on the system content and structure rather than on structuring equations for a particular causal representation of the system,

- This approach leads to the creation of hierarchical model libraries,
- The hierarchical nature of the model libraries promotes both model reuse and flexibility in modifying or creating new versions of the models in a reduced time frame,
- The object-oriented nature of the model library simplifies model maintenance by reducing the number of core models that need to be maintained.

Further, the ability to easily integrate model development into the overall system life cycle processes is integral in reducing the time, cost and effort involved in stakeholder communication, prototyping and design iterations. Specifically, the proposed method integrates well into the the systems engineering processes for:

- Requirements definition and refinement through early feedback of iterative development method, and
- Resource management through the early identification of modelling bottlenecks due to whole system view.

Referring to the aims and objectives stated in Section 1.3, the development method has been applied to the implementation of hierarchical libraries of reusable models. The use of which has allowed for the efficient evolution of a HEV powertrain design, having comparable results to an existing validated model.

# Chapter 6

# Case Study 2: LifeCar Extension

The automotive manufacturer Morgan intended using the original LifeCar project as a prototype with the aim of building on lessons learnt and producing a production ready lightweight HEV design. For the investigation presented in Chapter 5, which was carried out after the actual prototype development had been done, a resource of prior models, data and experience was available to aid in the powertrain modelling task. In contrast, this extension to the LifeCar project was an actual development project which started in the final year of the author's PhD period. This created an opportunity for the modelling method to be used in the initial stages of the design and contribute to the development of the energy storage system for this vehicle [168].

As with the development of the new car being based on lessons learnt from the prototype LifeCar vehicle, so the new models are developed by reusing and extending the library of packages and subsystems produced in Chapter 5 in order to meet the requirements for the new vehicle. In this chapter a second case study is presented, focusing on the changes and additions made to the HEV modelling library discussed in Chapter 5. The idea being to test the suitability of the modelling method in a situation where existing model designs and packages are being reviewed and structurally modified.

## 6.1 Requirements Analysis

Once again, the key factor in the initial stages of the design process described in Section 4.4, is communication between all parties involved in the design effort. During a stakeholder meeting at the initiation of the new project, the producers of the LifeCar vehicle stipulated that the new vehicle will take the form of a series HEV with the fuel cell power supply being replaced by an ICE/generator combination and a battery. It was further stipulated that the voltage bus should no longer be floating with an ultracapacitor directly coupled to it, but should be fixed at a constant value of 400 V. The option of adding an ultracapacitor for capturing braking energy, with a converter between the ultracapacitor and the high voltage bus to provide more control, forms part of this investigation. The scope of this study is firstly to examine the electric vehicle (EV) range and energy requirements for the LifeCar powertrain when used in an "electric only" mode while driving in an urban environment, and secondly to assess the acceleration performance provided by the power supply in that mode.

Since this study focuses on an "electric only" driving mode, the ICE and generator are not modelled but are considered as part of the vehicle mass. More specifically this study involves a comparison of EV powertrains supplied by; a battery alone, or dual source supply of a coupled battery and ultracapacitor. This comparison is performed through sets of test models for examining the vehicle range and vehicle performance as described by the following considerations for the modelling process:

A Electric Vehicle Range

1. Battery Only Power Supply
   - Account for energy losses in the system.
   - Incorporate regenerative braking strategy.
   - Determine the range provided by battery.
   - Understand the trade-off between battery size and range.
2. Dual Source Power Supply
   - Determine the size of ultracapacitor needed to capture lost braking energy.
   - Account for energy losses in the system.

   - Incorporate powersplit and regenerative braking strategies.
   - Determine the range provided by hybrid battery and ultracapacitor power supply.

3. Compare 4WD and 2WD Topologies

B Electric Vehicle Performance

1. Battery Only Power Supply
   - Replace chassis and tyre models with high fidelity models.
   - Determine the maximum vehicle acceleration.
2. Dual Source Power Supply
   - Replace chassis and tyre models with high fidelity models.
   - Determine the maximum vehicle acceleration.
3. Compare 4WD and 2WD Topologies

### 6.1.1 Initial Design Constraints

Manufacturer stipulated design targets for the new vehicle form part of the design boundaries for this exercise. The initial design constraints most relevant to this investigation are listed in Table 6.1.

Table 6.1: Initial design targets for LifeCar redesign.

| **Type** | **Value** | **[unit]** |
|---|---|---|
| Top speed | 185 | [km/h] |
| Electric only urban range | 30 | [km] |
| Acceleration (0–100 km/h) | 6.2–6.5 | [s] |
| Maximum vehicle unladen mass | 880 | [kg] |
| Maximum vehicle laden mass | 1100 | [kg] |
| Vehicle bus voltage (fixed) | 400 | [V] |

An example of a stakeholder initiated design change that needs to be considered when modelling the new powertrain, is that the new vehicle uses more powerful

electrical machines with respect to the original design. The specifications given by the electric motor designers for the new electrical machines are shown in Table 6.2.

Table 6.2: Permanent magnet electrical machine parameters.

| **Type** | **Value** | **[unit]** |
|---|---|---|
| Power rating | 100 | [kW] |
| Peak torque | 500 | [Nm] |
| Torque constant | 1.43 | [Nm/A] |
| Armature resistance | 0.03 | [Ω ] |
| Armature inductance | $325 \times 10^{-6}$ | [H] |
| Rotor inertia | 0.5625 | [kg.m$^2$] |
| Machine mass | 25 | [kg] |

As a starting point, the initial estimation of energy requirements and power supply size are chosen in order to meet the target range of 30 km of urban driving, together with an assumed laden vehicle mass based on initial mass estimates supplied by the vehicle manufacturer and other stakeholders. The mass breakdown used for this investigation is presented in Table 6.3. Since determining the mass of the power supply forms part of the investigation, it is not show in this table and is discussed later in Section 6.3.

## 6.2 Physical Modelling

For this case study, the main objective in the *Physical Representation* stage is to reuse and add to the library of subsystems and partial models developed for the original LifeCar case study in order to be able to develop and test the new powertrain design. Doing this provides a method for testing; the reusability of the previously developed library models, the suitability of the partial models for developing new model versions and the overall adaptability of the object-oriented model design for exploring and executing design changes. To achieve this, a new package of powertrain models is produced within the HEV modelling library constructed for the initial case

Table 6.3: Mass breakdown for new LifeCar vehicle.

| **Breakdown** | **Unladen [kg]** | **Laden [kg]** |
|---:|:---:|---:|
| Rolling chassis | 400 | 400 |
| Electrical machines (x4) | 100 | 100 |
| Engine | 150 | 150 |
| Generator | 25 | 25 |
| Cooling | 25 | 25 |
| Electronics | 50 | 50 |
| Power Supply | ?? | ?? |
| Driver and passenger | 0 | 150 |
| **Total** | **750** | **900** |

study. Within this new package, two further packages *EVRange* and *EVPerformance* are constructed as show in Figure 6.1. This is done to group models with similar purpose and fidelity in the same way that the *EnergyManagement* and *HiFidelity* packages were used in Chapter 5. Each package consists of a model for investigating a battery as the sole power source and another model for investigating a dual source power supply. It is worth noting that this does not imply a duplication of developed models since any newly created subsystems are added to the model library within the appropriate base subsystem package as shown in Figure 5.12. The powertrain models then simply inherit a common base vehicle model and replace subsystem models as appropriate. In this way, any changes made to the models in the subsystem package will be effected on all powertrain models using that subsystem.

### 6.2.1 Hierarchical Abstraction and Partial Models

As mentioned in Section 4.2.2, the steps in an object-oriented design don't need to be carried out in a strictly sequential manner, thereby allowing tasks to proceed without needing a prior step to be fully completed. In other words, an iterative cycle can generally begin wherever is easiest within the cycle as long as the cycle is then fully iterated. In this case the subsystem level hierarchical abstraction from the design in Chapter 5 can still be considered valid since the focus of this study is on the "electric only" capabilities of the new LifeCar architecture. This indicates that the previously developed library structure and partial models should be well suited

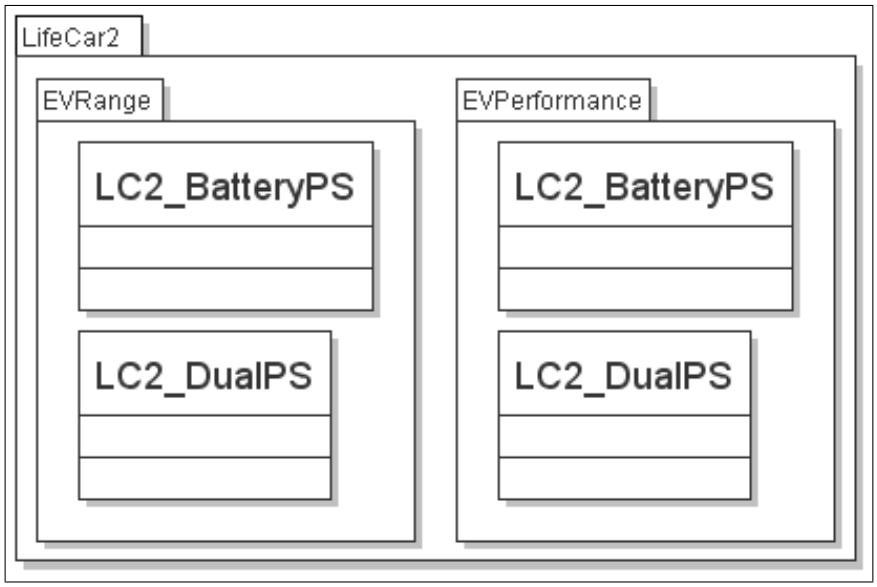



Figure 6.1: Package structure for new LifeCar powertrain models.

for the development of the new powertrain subsystem. Thus, further abstraction and partial model development only need occur where a new subsystem specific to the new architecture must be added to the model library. To maximize the reuse of the existing library and reduce the modelling effort going forward, the developer should focus on the differences in the model structure and architecture by taking into account factors such as:

1. The model structure must allow for optional two wheel drive powertrain designs.
2. The model structure must allow for investigating both power supply options.

An example of how a suitably generic partial model can be reused in several different instances, is seen in the partial vehicle model developed for the first case study which is shown in Figure 5.3. Since this model does not specify any power supply requirements and only enforces one driveline subsystem, it is suitably generic for power supply investigations on both two wheel drive (2WD) and four wheel drive (4WD) architectures. Therefore, for these preliminary investigations, no further abstraction is necessary at this level. At a later stage in the design process, the abstraction must be revisited when the final architecture of the new design is investigated. Fritzson [58] suggests that the ability to find a balance between model simplicity and preciseness is more of an art than a science which is acquired through experience. Further abstractions could then include the addition of the combustion engine and generator as well as possible additions to the driveline architecture if,

for example, it is decided to investigate the use of a differential in order to reduce the number of electrical machines.

### Partial Models

In Chapter 5 a converter from the SED library was used in the modular power supply shown in Figure 5.9. However, this converter model did not account for any energy losses and did not allow for two power supply sources and converters to be connected in parallel due to the manner that the power balance between input and output side is implemented. Additionally, in order to implement the two power supplies in parallel it is also necessary to be able to control what portion of the demanded power would be attributed to each supply. Therefore, a new partial model for power electronic converters is constructed and added to the *PartialModels* package in the HEV modelling library. As shown in Figure 6.2, this partial model defines the connections to an electrical supply and an electrical load. The same partial model can also be extended with an additional set of supply ports in order to construct a dual input converter needed for the dual source power supply.


Figure 6.2: Partial model for a power electronic converter

### 6.2.2 LifeCar Subsystem Changes

At this point it is possible to commence with the fifth step in the proposed modelling method. This section discusses the subsystems used in the new LifeCar model that differ from those already described in Chapter 5. In particular, models for the power supplies, converters, power split controllers and braking controllers.

#### Batteries and Ultracapacitors

For this investigation a new battery model and ultracapacitor model are added to the *PowerSupplies* subsystem package. Once again, in order to investigate the advantages and disadvantages of off-the-shelf component libraries, pre-built components from the SED library were selected. Both the battery and ultracapacitor models selected represent the battery pack as a combination of series and parallel cells where the number of cells, maximum cell voltage, maximum cell current, internal cell resistance and initial state of charge (SOC) can be defined. Additionally the battery model requires a maximum and minimum SOC value to be specified in order to track the instantaneous SOC as the voltage varies, and to terminate the simulation with a warning message when the minimum SOC is reached. Both models are based on ideal first order models with a capacitance and series resistance for each cell in the pack. The battery model uses a capacitor instead of a voltage source in order to better model the change in SOC [169].

Both these components were not used directly in the model but were placed inside the previously constructed (shown in Figure 5.5) partial model for power supplies. This is done in order to provide a connection to the ControlBus and to allow the models to be easily replaced by any new power supply models constructed in the future. Real world data is then used to parameterize these models. More specifically, with regards to the particular battery and ultracapacitor technologies consulted for this exercise, Lithium Iron Phosphate ($LiFePO_4$) battery cells and `Boostcap`® ultracapacitor cells were chosen. $LiFePO_4$ are used due to the high discharge current rating of this battery technology and the Maxwell Technologies are used due to existing stakeholder experience with this technology in the initial LifeCar design. Specific cell data used is shown in Tables 6.4 and 6.5.

Table 6.4: LiFeBATT$^{TM}$ battery cell specifications.

| **Battery** | **Value** | **[unit]** |
|---|---|---|
| Typical Cell Voltage | 3.3 | [V] |
| Cell Capacity | 8 | [Ah] |
| Max. Energy per Cell | 24 | [Wh] |
| Internal Cell Impedance (initial) | 0.006 | [Ω ] |
| Max. Discharge Current (25C) | 200 | [A] |
| Max. Charge Current (4C) | 32 | [A] |
| Cell Mass | 0.29 | [kg] |
| Energy Density | 80 | [Wh/kg] |
| Power Density | 2000 | [W/kg] |

Table 6.5: Maxwell Technologies® ultracapacitor cell specifications.

| **Ultracapacitor** | **Value** | **[unit]** |
|---|---|---|
| Cell Capacitance | 3000 | [F] |
| Volume | 0.414 | [l] |
| Mass | 0.55 | [kg] |
| Equivalent Series Resistance | $2.90\text{x}10^{-4}$ | [Ω ] |
| Max. Cell Voltage | 2.7 | [V] |
| Max. Continuous Current | 150 | [A] |
| Energy Density | 5.52 | [Wh/kg] |
| Power Density | 5400 | [W/kg] |

**Power Electronics Bidirectional Converters**

As mentioned earlier a new converter model was required in order to account for the converter energy losses. These are determined by considering the switching ($P_{sw}$) and conduction ($P_{cond}$) power losses in the converter's insulated-gate bipolar transistors (IGBTs) as described in equations (6.1)–(6.3). Switching losses are calculated as a function of the the switching current ($I_{sw}$) for a specific switching frequency ($F_{sw}$), this function was implemented in the model as a look-up table of losses per cycle taken from manufacturer data. The conduction losses are estimated as the

power dissipated due to the constant voltage drop across the IGBT ($V_{IGBT}$).

$$P_{sw} = F_{sw}\ f(|I_{sw}|) \tag{6.1}$$

$$P_{cond} = |I_{supply}|\ V_{IGBT} \tag{6.2}$$

$$P_{convLoss} = P_{sw} + P_{cond} \tag{6.3}$$

where: $I_{supply}$ is the current delivered by the power supply and $P_{convLoss}$ is the total converter power losses.

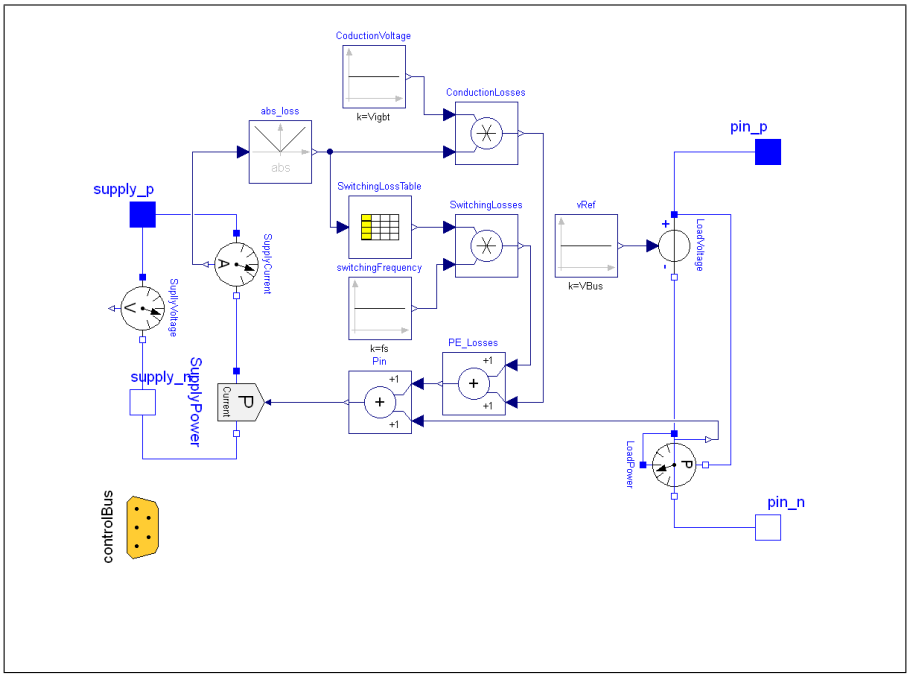



Figure 6.3: Power electronics converter with switching and conduction losses.

The new converter model then adds the total converter losses to the load power in order to calculate the required power at the supply side of the boost converter as shown in Figure 6.3. The voltage at the load side of the converter is set to keep the required constant bus voltage and the supply current is determined by balancing the supply power with the sum of the load power and converter losses.

In order to implement a dual power supply, a new converter model was constructed with a second set of supply connectors as shown in Figure 6.4(a). Since the additional supply port essentially represents a second converter, the inverter losses are calculated in the same way for the second supply. Further, an idealized energy management strategy to split the load power between the two power sources is included in order to perform the initial tests.

The control logic used to implement this strategy is developed as a separate model class as shown in Figure 6.4(b), which is added within a controller sub-package of the converter base model package in the HEV library hierarchy. An instance of this model is then included within the converter model as a replaceable controller object. In this way, the fact that this converter requires a controller is clearly visible to future users of the model and more importantly, the energy management strategy can be easily updated and replaced in future applications by simply creating adding new controller model to the library.

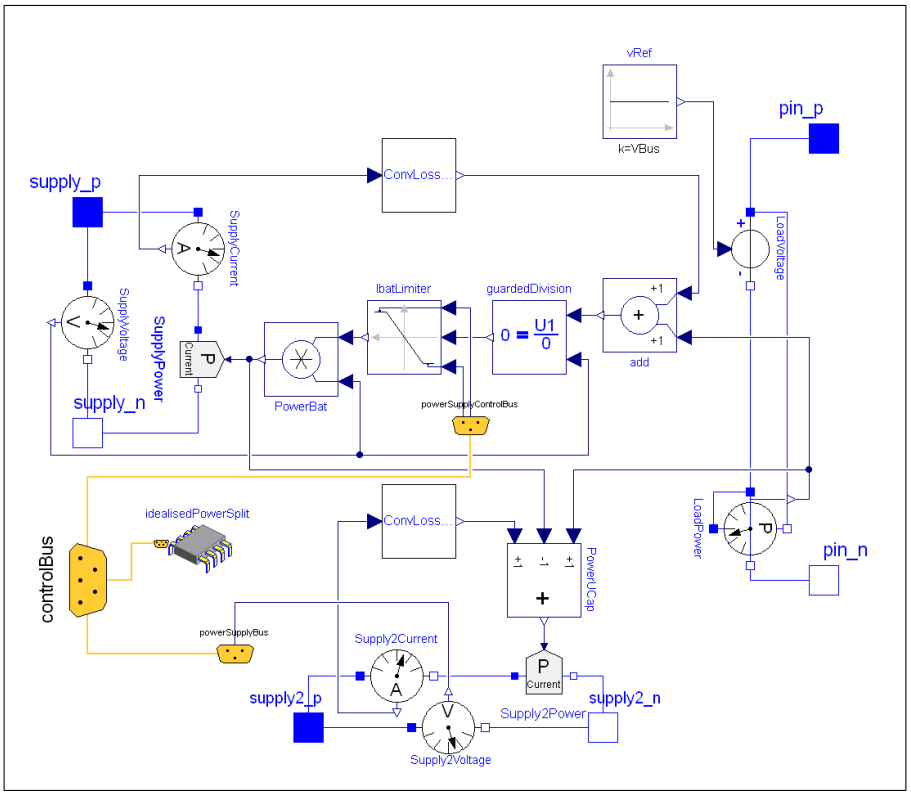
Since this investigation is performed primarily as a proof-of-concept, the idealized energy management strategy makes use of a rather simplistic control logic to perform the power split. The idealized control strategy is based on a hysteretic charging and discharging cycle of the ultracapacitor according to the following rules:

1. While ultracapacitor is charging from minimum SOC to maximum SOC.
    - Use the battery as the main source for powering the vehicle.
    - Use the ultracapacitor as the main storage when braking.
2. While ultracapacitor is discharging from maximum SOC to minimum SOC.
    - Use the ultracapacitor as the main source for powering the vehicle.
    - Use the battery as the main storage when braking.

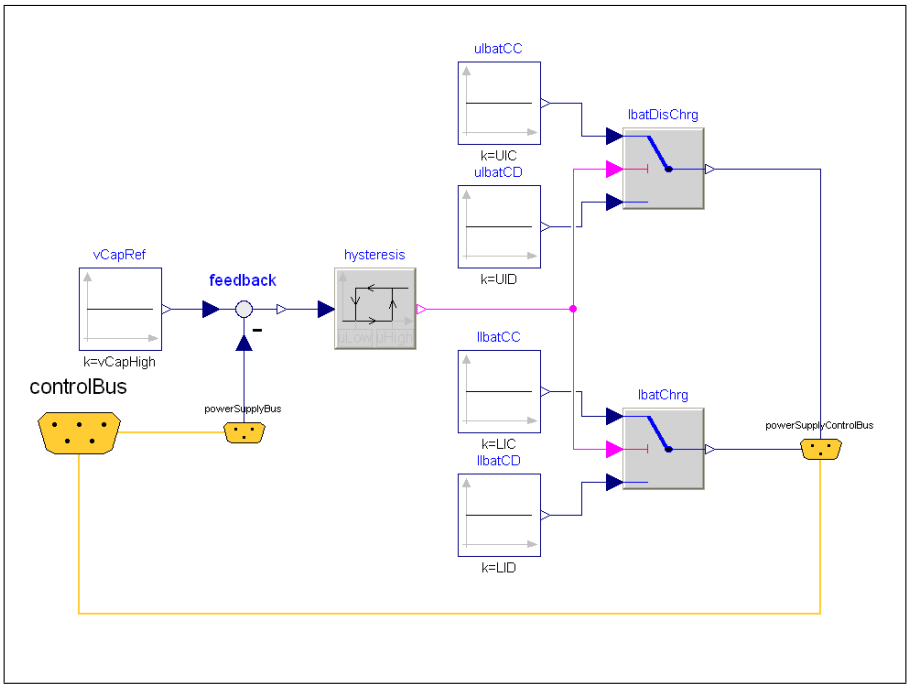
The hysteresis block seen in Figure is used to ensure the a cyclic charging and discharging of the ultracapacitor between maximum SOC and minimum SOC limits. This is used as a simplified manner of avoiding the phenomenon of chattering when the ultracapacitor approaches a the border between charging and discharging states [170].

### Brakes Controller

One of the major implementation differences between the first and second case studies is the inclusion of energy management control functions within the developed models. For the case study in Chapter 5, the energy management functions were only provided in Simulink and there was no regenerative braking implemented in the Dymola model. Further, the vehicle brake balance was assumed to be evenly distributed between front and rear wheels. For this case study, in order to size the




(a) Dual input converter




(b) Power split controller

Figure 6.4: Dual supply power electronics converter with idealized energy management strategy.

power supplies, examine the vehicle range and compare 2WD and 4WD powertrain options, a regenerative braking strategy is necessary. Therefore, a brake controller is added to the base subsystem library. Figure 6.5 shows how the logic for the regenerative braking strategy is implemented by this controller.

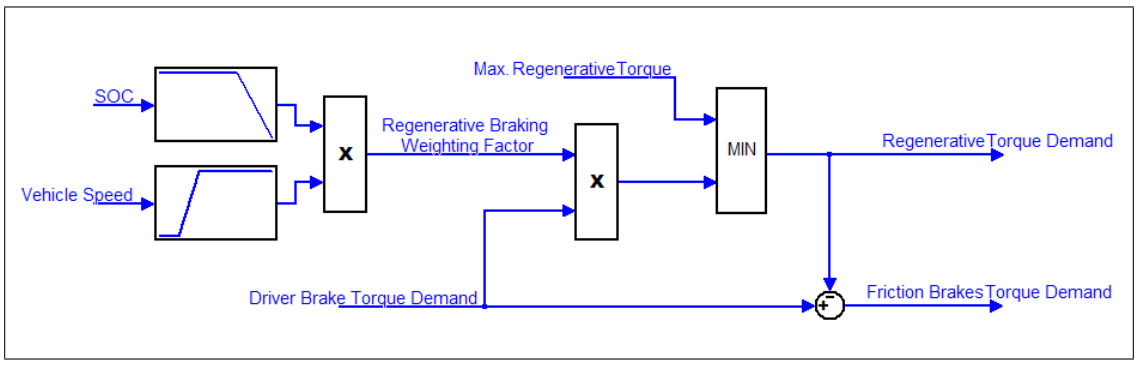



Figure 6.5: Regenerative braking strategy.

The controller uses the vehicle speed and battery SOC to determine a weighting factor for regenerative braking ($k_{regen}$) where; no regeneration is possible at speeds lower than $4\,m.s^{-1}$ and battery SOC above 80%, and full regeneration is allowed at speeds above $8\,m.s^{-1}$ and SOC below 70%. As was explained in Chapter 5, this is done because of the low potential to store energy and low generation efficiency at low speeds. This weighting factor is then used to scale the total driver demanded braking torque ($\tau_{dmd}$) so as to determine the balance between the regenerative machine torque request ($\tau_{req}$) and the friction torque according to equation (6.4).

$$\tau_{req} = k_{regen}\ \tau_{dmd} \tag{6.4}$$

In order to determine the actual torque demand ($\tau_{regen}$) that can be passed on to the electrical machines, the requested regenerative torque is compared with the maximum regenerative torque ($\tau_{max}$). This maximum torque is dependent on the maximum power that the power supply can accept during regenerative braking and is calculated as described by equations (6.5) and (6.6).

$$\tau_{max} = \frac{V_{bat}\ I_{charge}}{\omega} \tag{6.5}$$

$$\tau_{regen} = \mathrm{MIN}\left(\tau_{req}, \tau_{max}\right) \tag{6.6}$$

where: $V_{bat}$ is the battery pack voltage, $I_{charge}$ is the maximum charging current of 4C for the chosen battery technology and $\omega$ is the angular velocity of the wheel and motor shaft.

Finally, the remainder of the driver demanded braking torque that cannot be supplied by the electrical machines constitutes the torque demand for the friction brakes ($\tau_{friction}$).

$$\tau_{friction} = \tau_{dmd} - \tau_{regen} \tag{6.7}$$

A user specified longitudinal brake balance is used to proportion the torque demands between the front and rear axles. These axle demands are in turn distributed equally between the left and right wheels since each wheel is powered by an individual electrical machine.

The same regenerative braking strategy is followed for the dual power supply with battery and ultracapacitor. The only differences being:

1. the ultracapacitor SOC forms part of the product defining the weighting factor, and
2. the ultracapacitor's energy storage capacity is added to that of the battery before calculating the maximum acceptable regenerative torque for the power supplies as described by equation (6.10) in Section 6.3.5.

### 6.2.3 New Powertrain Base Models

The final step of the *Physical Representation* stage in the proposed modelling method requires the HEV base model to be defined. This can be regarded as a second iterative spiral in the development method where the models developed in the first case study are treated as prototype iterations as described by the Spiral Model in Figure 4.2. It is important that the development method allows for the both the reuse and modification of previous development effort while not restricting the developers flexibility by enforcing prior design decisions which may no longer be applicable. Specifically, in this case, the new powertrain base model must allow for modifications to the original design that not only require changing component fidelity by replacing existing components with different version, but also require architectural changes such as removing components completely. For this reason it is not appropriate to extend or inherit the previous LifeCar base model, since inherited models only permit addition or replacement of components but not removal. A better solution is to create a new base model by reusing and extending the more generic partial

vehicle model shown in Figure 5.3, and thereby cater for the differences in the model structure mentioned previously in Section 6.2.1.

As mentioned in Section 6.1, the models developed in this case study are to be used for testing both the vehicle's range and performance for the different powertrain options. Since these tests would require different power supply and driver models and it was not known initially whether the driver functionality would be developed in the Dymola environment, it was decided not to include any driver or power supply subsystems in the new LifeCar base powertrain model. This had the benefit of allowing the power supply model development to occur independently and the plant model not enforcing any architectural limitations on this development. In other words the final vehicle plant model is developed by adding any driver and power supply options to the base model. Additionally, if the power supply options have already been developed in another modelling environment such as Simulink, this base model can exported and more easily integrated with existing models.

A 2WD base model as shown in Figure 6.6 is then defined by adding two electrical machines, a LifeCar driveline and a controlled brake subsystem from the created HEV model library. This base can be used for both front and rear wheel drive models by simply replacing a rear wheel driveline model with a front wheel driveline model. In order to produce a base for the 4WD powertrain models, the 2WD base model is inherited and extended with an additional driveline and two more electrical machines.

## 6.3 Executable Models and Simulations

For this case study, the final or *Implementation* stage of the modelling method is iterated several times in order to develop and test executable models for the various powertrain options being investigated. More specifically, a set of models is developed for drive cycle simulations in order to assess and compare the vehicle range when powered by the different power supply designs. These models are then modified with a higher fidelity chassis subsystem and tip-in tip-out simulations are performed in order to compare the vehicle performance in terms of the maximum achievable acceleration when powered by the different power supply options. Finally, the effect and feasibility of reducing the vehicle mass and cost by removing two of

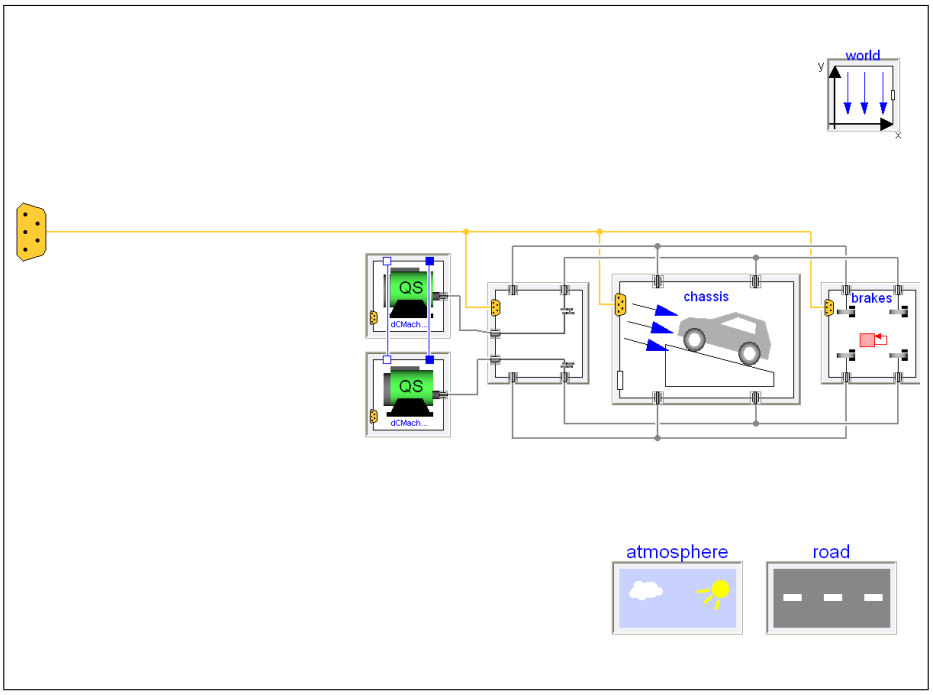



Figure 6.6: Base model used for creating EV range and performance powertrain models for the LifeCar extension.

the electrical machines is investigated by performing the same tests on both front and rear 2WD powertrain models.

Vehicle range and performance simulations are presented first for the battery powered models and then for the dual source powered models. In order for any of the executable models to be correctly parameterized, the size of battery pack that would allow the LifeCar vehicle to cover a distance of 30 km was estimated according to the net energy required to cover this distance. Energy usage is affected by several external factors such as road type, traffic congestion and driving style [171]. Considering that there are many possible ways that a vehicle can cover a certain distance, with different speed and acceleration, a set of three urban drive cycles was used to produce an initial estimate of the energy demands.

These three drive cycles are shown in Figure 6.7, where the first two are legislative urban cycles which are commonly used for assessing vehicle fuel consumption and emissions via dynamometer tests. As mentioned in Section 5.3.2, the ECE is a stylistic modal cycle which has been designed for better understanding during analysis and the UDDS is a more realistic transient cycle. Even though the UDDS drive cycle is more realistic than the ECE cycle, it is still an artificially constructed

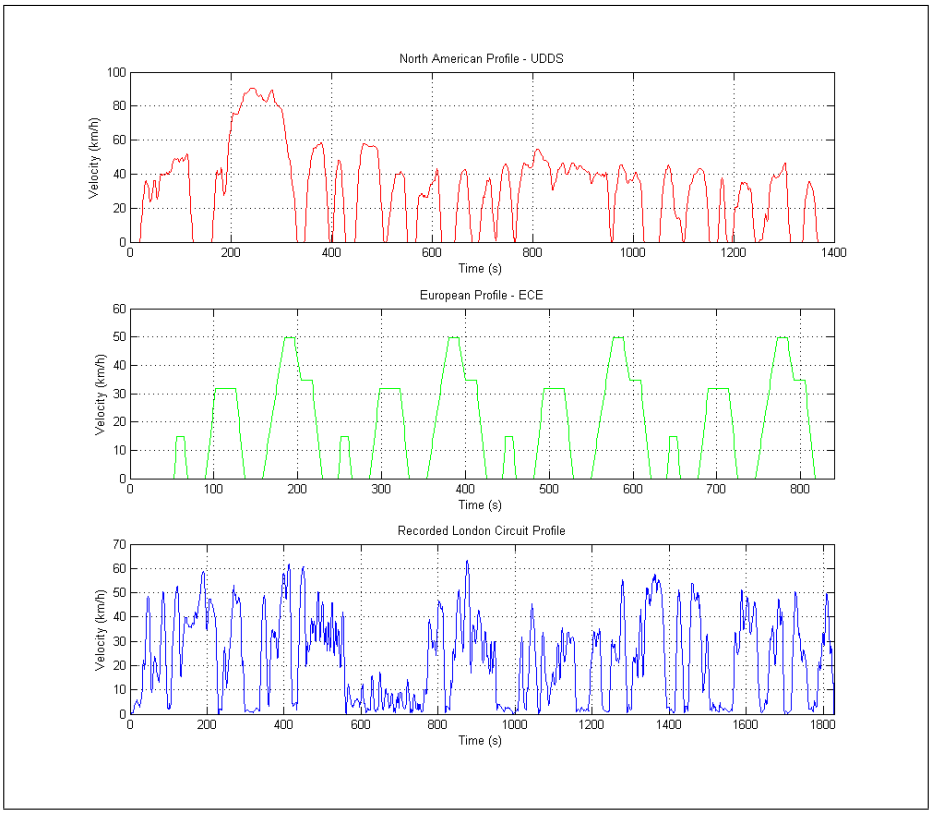



Figure 6.7: North American, European and recorded London drive cycles used for initial energy requirements test.

cycle designed for testing and analysis. Further, the UDDS cycle has an extended period at high speed which raises the average speed but is not typical for most congested cities where zero emission driving is called for. Therefore, in order to get a better estimation of the actual energy requirements for the EV mode testing, a real world cycle recorded while driving in the city of London is also used for the simulation tests. The London drive cycle represents a moderate driving style in an urban environment.

A comparison of the net energy requirements for the different drive cycles is presented in Table 6.6. Here it can be seen that the net energy requirements for the vehicle to cover 30 km under the transient cycle are 40% larger than those for the modal cycle, and requirements for the real world cycle are a further 12% greater. In order to arrive at an estimated battery pack size and mass, a mass sensitivity study consisting of multiple simulation iterations was performed varying the number of cells in the battery pack and the corresponding change in vehicle mass was performed. This study revealed an increase in energy requirements of 2.7 Wh/kg and led to a 40 kg power supply being chosen to meet the simulated requirements for the 30 km

range. Remembering that as a starting point, these initial net energy values are independent of any particular battery technology and assume that the power supply is able to recapture all the braking energy provided by the specific drive cycle.

Table 6.6: Comparative energy requirements over 30 km range (940 kg vehicle).

| | **Drive Cycle** | | |
|---|---|---|---|
| | **ECE** | **UDDS** | **London** |
| Max. speed [km/h] | 49.7 | 90.7 | 63.4 |
| Avg. speed [km/h] | 18 | 31.3 | 22.5 |
| Simulated drive time [min] | 100 | 55.5 | 80 |
| Net Energy [kWh] | 1.16 | 1.62 | 1.81 |

For this study it is assumed that the electric vehicle mode is required mainly when in urban areas, therefore the recorded city drive cycle is used for the remainder of the investigation. Using the calculated net energy requirement for the London drive cycle, the final battery size is found by considering the $LiFePO_4$ battery's charging limitations and assuming that the energy is provided by 80% of the batteries capacity. A useful capacity of 80% is assumed in order to leave a safety buffer to prevent either overcharging or completely discharging the battery and also to allow the battery to be operated in a region where its voltage characteristics are more stable. This depth of discharge is specific for EV mode operation as the battery is the only power source, during HEV operation this would be reduced to around 60% or less to increase the battery cycle life [165].

Figure 6.8 shows the current and power demands corresponding to the London drive cycle data presented in column 3 of Table 6.6. In the second plot it can be seen that the demanded supply current often exceeds the 32 A recharge limit for the chosen battery type. In order to prevent these high braking currents, the excess braking energy would have to be dissipated in the friction brakes. The bottom plot shows the power demand on the battery (blue) and the lost power due to the recharge current limitations (red). Also, noting that the net energy for this 30 km profile is 1.81 kWh (green) and the uncaptured energy is 0.25 kWh (red dotted), the required battery energy is 2.06 kWh (blue dotted).

Since this energy comes from 80% of the battery capacity, the battery is sized for

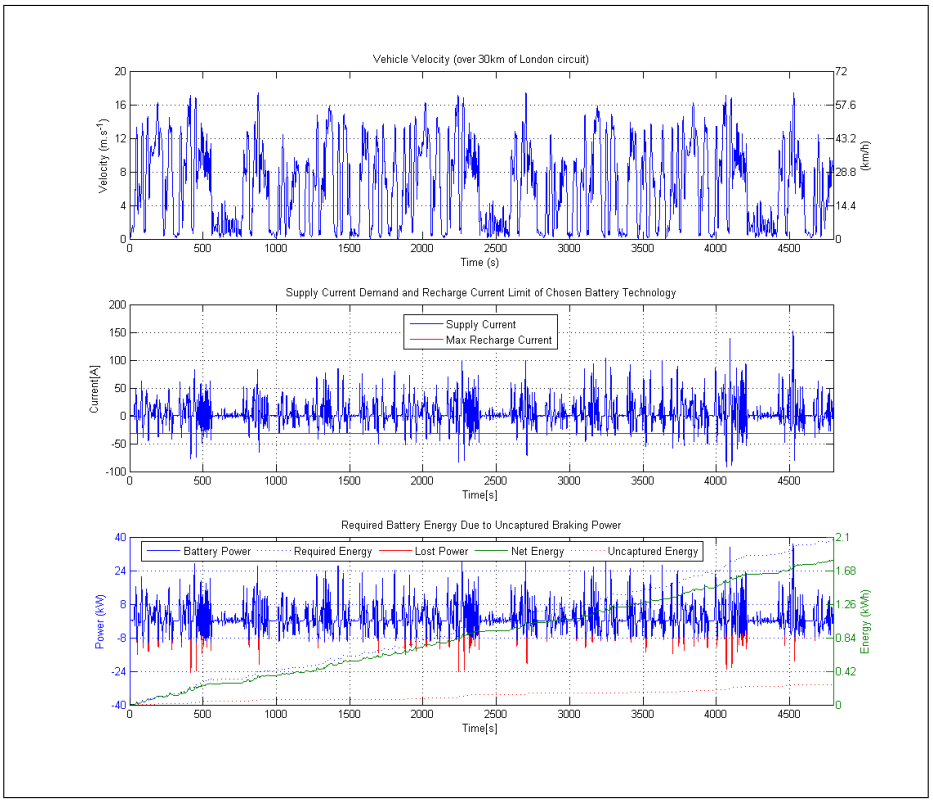



Figure 6.8: Required battery energy taking charging current into account.

2.5 kWh. Using the $LiFePO_4$ manufacturer data from Table 6.4, this equates to a 356 V battery pack of 108 cells in series. A parrallel arrangement of battery cells can be used in order to increase the current capacity of the battery but this was not considered since the 25C current rating for this battery technology allowed for sufficiently high currents of up to 200 A. The total pack mass for the battery and packaging is 40 kg, which when added to the mass breakdown shown in Table 6.3, gives a total laden vehicle mass of 940 kg for the battery powered model.

### 6.3.1 Battery Powered EV Range Model

At this stage of the development, there are sufficient additions and modifications to the HEV model library in order to create the visually representative powertrain models as was done in Chapter 5. The executable model for the battery powered EV range model is created by inheriting the newly defined 4WD base model and connecting the battery model via a single supply converter model to the electrical machines. A drive cycle driver model is included and the new regenerative braking controller model is added to the brake model.

As with the original LifeCar model shown in Figure 5.15, the electrical machines require a torque reference which in this case is produced by combining the driving torque and regenerative braking torque demands. This combination is done externally at the top level of the model hierarchy, since it is likely that the driver functionality, the energy management and regenerative braking strategies will change during the initial development stages. At a later stage, once experience has been gained, this functionality can be incorporated into a purpose built driver model for example, and an appropriate level of controller replaceability defined. In the same way, the battery and converter models are not combined into a single power supply model since the power supply is the main subsystem being investigated. Also, it is not possible to replace the single input converter with the dual input version since they require a different amount of connectors.

Figure 6.9 shows the complete executable model used for the battery powered range investigation. It is important to understand that up to this point the development method has allowed the developer to focus on the subsystems which form the main contribution to the study. Namely, development effort is focused on making additions to the HEV model library for the power supply, converter and regenerative braking strategy. Changes to the base model involved setting the level of abstraction to a 2WD base instead of a 4WD base but did not require much development effort since the originally defined partial model abstraction was reused. All other subsystems in the executable model are carried forward from the study in Chapter 5 in the same manner as prototype models are refined within each cycle of the Spiral development model.

### 6.3.2 Battery Powered Model: EV Range Simulations

Initially the battery was sized in order to provide power for a 30 km range with the London drive cycle, but this assumed all braking energy could be recaptured. With the regenerative braking strategy in place, the battery is no longer able to power the vehicle for this distance due to the implementation of the weighting factor. Further, the 4C charging current limit of the battery also limits the regenerative power to just over 11 kW without taking the battery power losses into account. Simulations are started at 90% SOC as the manufacturer of the battery stated that the battery performance is more stable in the middle 60% of the operating range and it is therefore assumed that the battery will not be recharged to 100%. Figure 6.10(a)

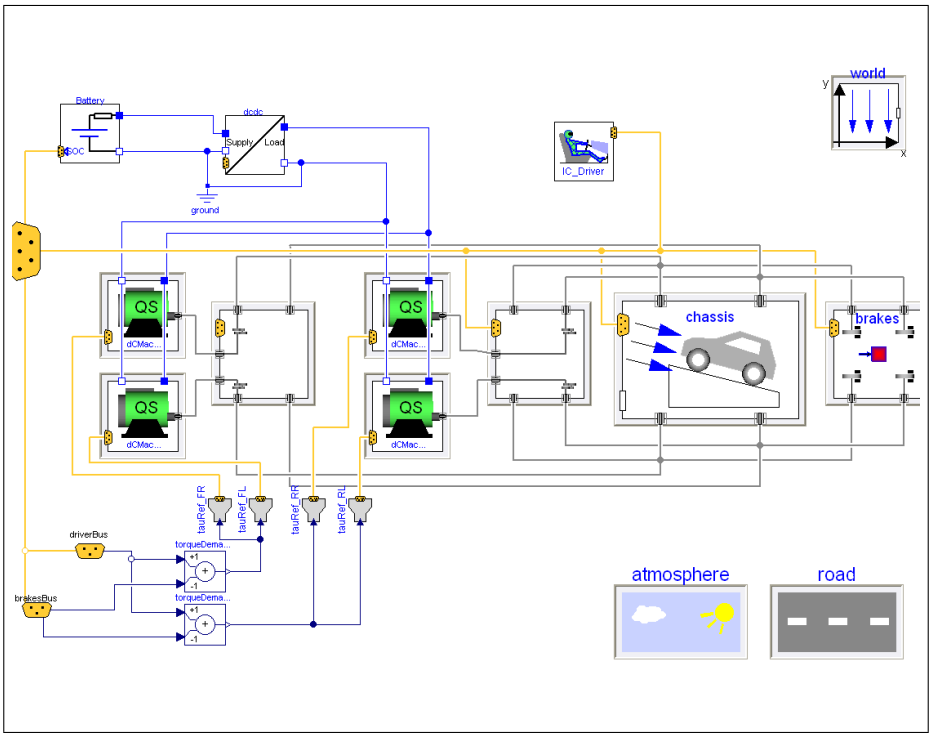


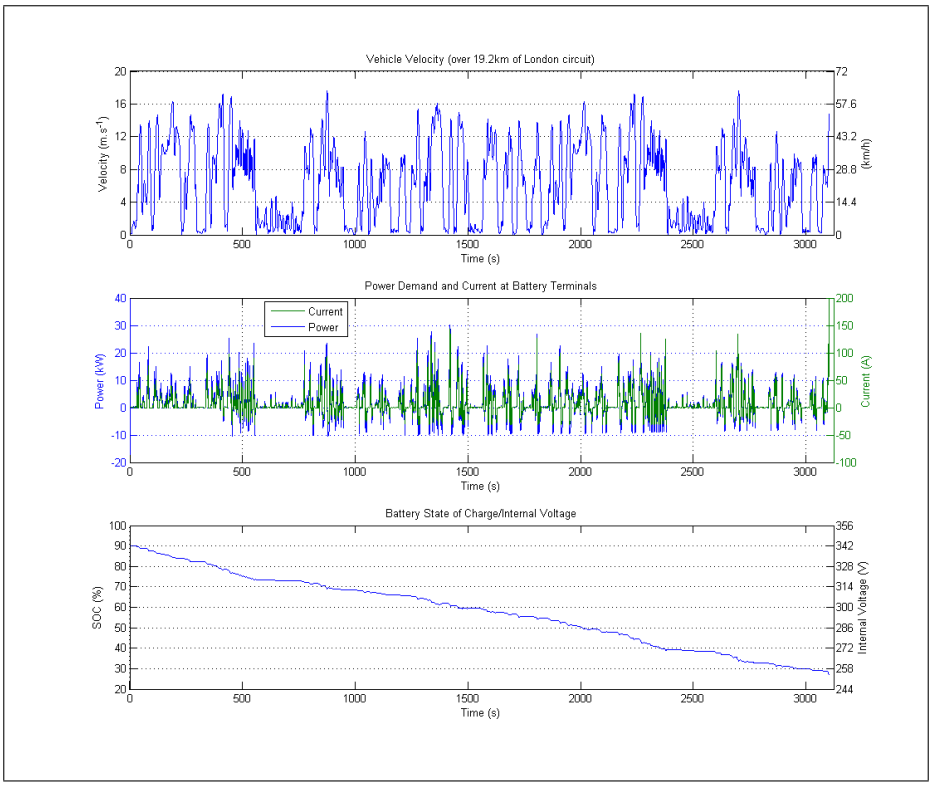

Figure 6.9: Battery powered powertrain model for LifeCar EV range investigation.

shows the power and current demand at the battery terminals during a recursive run of the London driving cycle.

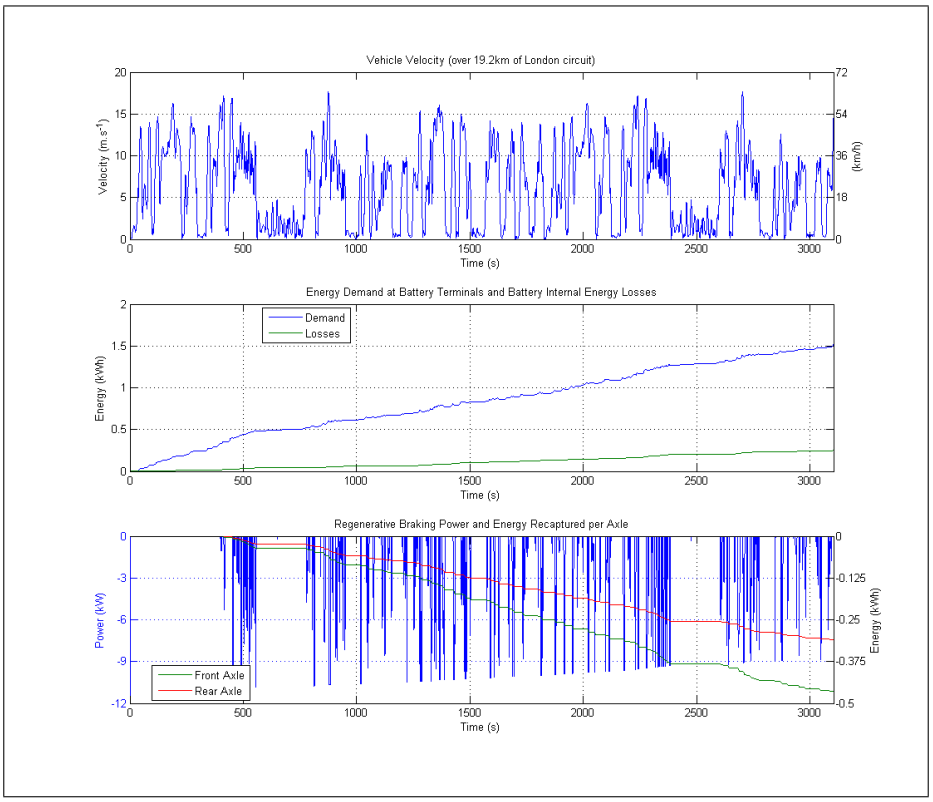
After 19.2 km, the simulation terminates due to the battery current reaching the maximum discharge limit of 25C or an upper current limit of 200 A when the SOC reaches 27%. In the implemented battery model, the SOC is based on the internal voltage ($V_b$) and the terminal voltage ($V_t$) is calculated by subtracting the voltage drop over the internal resistance ($IR_b$) according to equation (6.8).

$$V_t = V_b - IR_b \tag{6.8}$$

Considering that the battery pack consists of 108 series battery cells with a constant internal resistance of 6 m$\Omega$, at 200 A this voltage drop is 130 V and therefore the vehicle becomes power limited. This ideal battery model was based on a constant internal resistance due to the relatively constant voltage characteristics of the $LiFePO_4$ cells over stable operating range [172], and in order to acquire an initial comparison of the power supply options. Further, the value of internal resistance used corresponds to the maximum internal resistance for the $LiFePO_4$ cell, and was thus used as a worst case scenario.




(a) Battery Power, Current and SOC




(b) Energy Usage

Figure 6.10: Simulation results for battery powered range test.

The author recognizes that the results provided by this simplified battery model are not conclusive but they still have value in a comparative capacity and as proof of overall model functionality in the modelling process. Development of more accurate electrical models for the $LiFePO_4$ battery technology is the topic of current research [173]. As better models are developed, they can be easily integrated into the HEV library and replaced in existing powertrain models as an increased fidelity of the battery subsystem model.

Figure 6.10(b) shows that the energy demand at the battery terminals is 1.5 kWh and the energy lost in the battery due to $I^2R$ losses is 0.25 kWh giving a net energy usage of 1.75 kWh after only 19.2 km of the London drive cycle. As stated in Section 6.2.2 the regenerative braking weighting factor does not allow for regeneration at speeds below 4 $m.s^{-1}$ and while the SOC is above 80%, thereby reducing the amount of energy that can be recaptured. This can be seen clearly in the bottom plot of Figure 6.10(b) where the regenerative braking power is seen to be zero for the first 400 s while the battery SOC is above 80% and also has distinct bands of zero power whenever the speed drops below 4 $m.s^{-1}$. It must be remembered that the 80% SOC limit is a control variable used to determine the regenerative braking weighting factor and must be optimized and calibrated by the user during further control implementation studies. In practice it is likely that a different SOC control limit will be used depending on the whether the vehicle is operating in an EV or a HEV mode.

Also seen in the bottom plot of Figure 6.10(b), is the difference in recaptured energy at the front and rear axles due to the 60/40 brake balance implemented in the regenerative braking controller for this test. Negative energy values are used to denote the direction of energy flow as being into the battery and therefore the total energy recaptured from all wheels is 0.77 kWh.

### 6.3.3 Battery Powered EV Performance Model

The performance model makes use of the same powertrain model as the range model with the chassis subsystem replaced by a high fidelity chassis with weight transfer effects and non-linear tyre models as used in Section 5.3.3 of Chapter 5. Without knowing the final vehicle components and layout, the mass split between front and rear axles was estimated from the wheelbase and the centre of gravity data provided

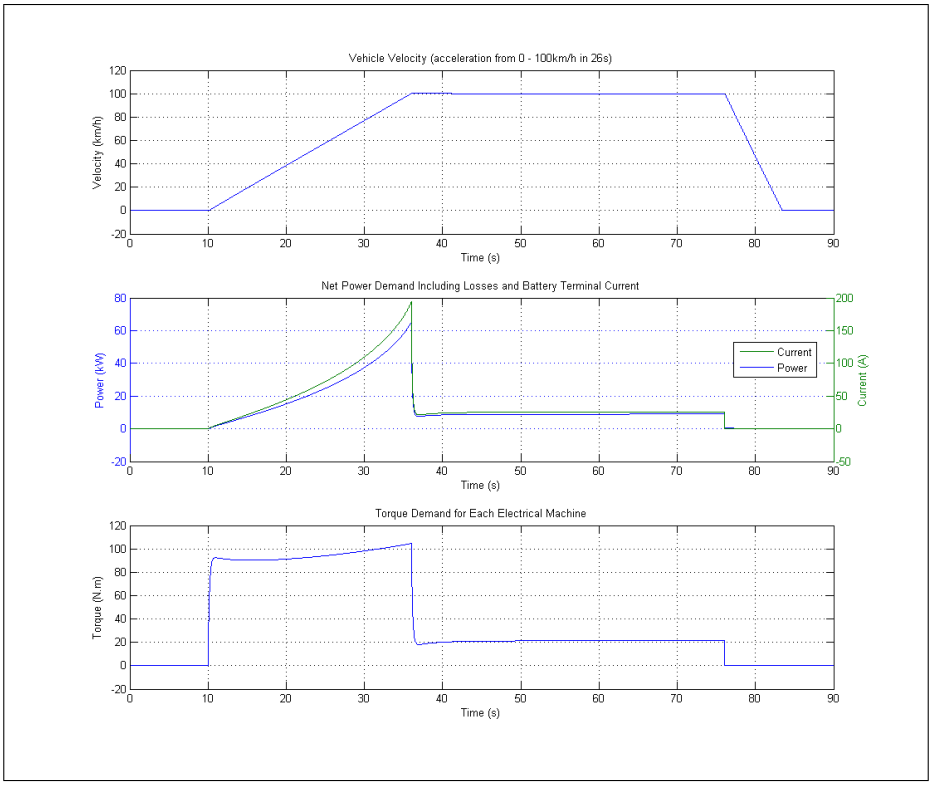
by the chassis manufacturer. This resulted in a mass distribution of 451.5 kg on the front axle and 488.5 kg on the rear axle. For test purposes the tyre model is parameterized with a dry road surface parameter set taken from Short et al. [161].

### 6.3.4 Battery Powered Model: EV Performance Simulations

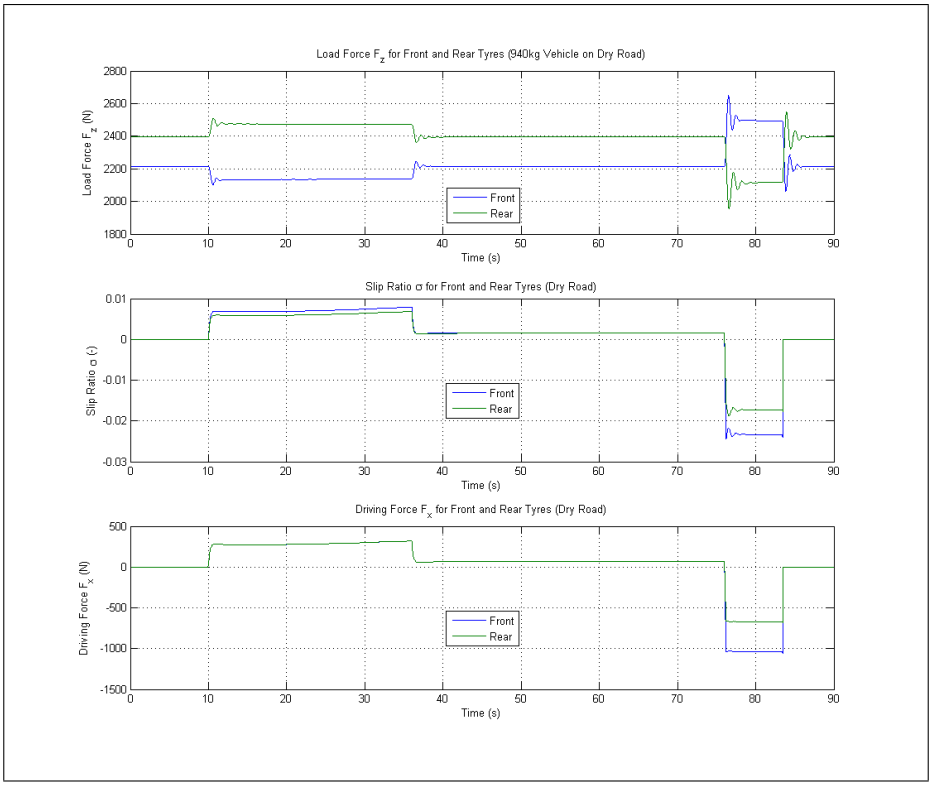
The intention of the performance simulation is to determine the maximum acceleration achievable by the vehicle in an "electric only" driving mode with the chosen battery size. For this test the London drive cycle was replaced with a tip-in tip-out speed profile with a maximum speed of 100 km/h. This maximum speed is used since it is slightly greater than the maximum speed of both North American and European legislative urban drive cycles.

Figure 6.11(a) shows the maximum acceleration achievable without power limiting the battery. As can be seen the peak power demand reached in accelerating the vehicle for 0-100 km/h in 26 s is over 65 kW. Considering that the battery starts at a 90% SOC (342 V), the maximum power it can supply at 200 A is 68.4 kW. Accelerating any faster would place a higher torque demand on the electrical machines and hence increase the power demanded from the battery. There is no regenerative power during the braking manouver since the battery SOC is still above 80% at this point and the same charging control setpoints as for the vehicle range tests were used. As mentioned earlier, in practice these setpoints are optimized and different control logic is implemented during different vehicle operating modes.

Figure 6.11(b) shows the vehicle squat and pitch, with more weight transfer to the rear while accelerating and to the front while braking. The middle plot shows that the since there is less weight on the front wheels during acceleration but the same input torque to all the electrical machines, the front wheels tend to slip more than the rear wheels. Under braking the front wheels have a more negative slip ratio or tend to lock more, since the front brakes provide 60% of the braking torque while the rear brakes provide only 40%. This is confirmed in the bottom plot where the more negative driving force on the front wheels shows that the majority of the braking force occurs at the front wheels.




(a) Battery Power, Battery Current and Machine Torque Demands




(b) Load Force, Slip and Driving Force

Figure 6.11: Simulation results for battery powered performance test.

### 6.3.5 Battery and Ultracapacitor Powered EV Range Model

For the second set of test models, the converter model is replaced by the dual input converter model so that an ultracapacitor can be added to the vehicle power supply. The ultracapacitor is sized in order to store the 0.25 kWh of energy that could not be captured by the battery due to its recharging current limit as shown in Figure 6.8. Five types of ultracapacitor cells ranging from 650 F to 3000 F were considered as shown in Table 6.7. In each case, the number of ultracapacitor cells required is calculated according to the equation (6.9).

$$\begin{aligned} E_{uc} &= \frac{1}{2} C V_{max}^2 - \frac{1}{2} C V_{min}^2 \\ &= \frac{1}{2} \left( \frac{C_{cell}}{n_{cell}} \right) \left[ (n_{cell} V_{cell_{max}})^2 - V_{min}^2 \right] \end{aligned} \tag{6.9}$$

where: $E_{uc}$ is the required energy storage in the ultracapacitor of 0.25 kWh, $C_{cell}$ is the cell capacitance, $n_{cell}$ is the number of cells required, $V_{cell_{max}}$ is the maximum cell voltage of 2.7 V, and $V_{min}$ is the minimum allowable operating voltage for the ultracapacitor bank of 133 V. The value of $V_{min}$ is derived from a rule of thumb given by the converter supplier stating that a maximum converter boost ratio of 3 should be assumed in order to achieve the constant 400 V bus voltage for the LifeCar application.

Table 6.7: Sizing of ultacapacitor bank for $E_{uc} = 0.25$ kWh

| Cell Capacitance ($C_{cell}$) [F] | Number of Cells ($n_{cell}$) | String Mass [kg] | String Volume [l] | Max String Voltage ($V_{max}$) [V] | String Capacitance (C) [F] |
|---|---|---|---|---|---|
| 650 | 390 | 78 | 58.5 | 1053.0 | 1.7 |
| 1200 | 219 | 65 | 51.0 | 591.3 | 5.5 |
| 1500 | 180 | 57 | 47.5 | 486.0 | 8.3 |
| 2000 | 142 | 56 | 44.3 | 383.4 | 14.1 |
| 3000 | 107 | 58 | 44.3 | 288.9 | 28.0 |

Only the 2000 F and 3000 F options had a voltage below the desired 400 V bus voltage. Based on the smaller string size need, the 3000 F capacitor cell with lower series resistance and higher continuous current properties shown in Table 6.5 was chosen.

More specifically, an ultracapacitor bank with the characteristics shown in Table 6.8 is required in order to store the 0.25 kWh of uncaptured braking energy. Further, adding the 60 kg mass of the ultracapacitor bank to that of the 40 kg battery pack, gives a new power supply mass of 100 kg and a total laden vehicle mass of 1000 kg.

Table 6.8: 0.25 kWh Ultracapacitor bank properties.

| **Ultracapacitor** | **Value** | **[unit]** |
|---|---|---|
| Number of cells | 107 | [-] |
| Volume | 44.3 | [l] |
| Mass | 60 | [kg] |
| Equivalent Series Resistance | 0.031 | [Ω ] |
| Max. Voltage | 289 | [V] |
| Min. Voltage | 133 | [V] |
| Capacitance | 28.04 | [F] |

Finally, the regenerative braking control strategy is modified to take both power supplies into account. This is done by summing the maximum charging power of the battery and ultracapacitor in order to calculate the maximum allowable regenerative torque. To realize this change, equation (6.5) in Section 6.2.2 is replaced by equation (6.10).

$$\tau_{max} = \frac{(\mathrm{V}_{\max} - \mathrm{V}_{uc})\,\hat{\mathrm{I}}_{uc} + \mathrm{V}_{bat}\,\mathrm{I}_{charge}}{\omega} \tag{6.10}$$

where: $\mathrm{V}_{uc}$ is the ultracapacitor bank voltage and $\hat{\mathrm{I}}_{uc}$ is the maximum rated ultracapacitor current.

The complete executable model used for the dual source powered LifeCar investigations is shown in Figure 6.12. Here it can be seen that the vehicle base is the same as for the battery powered powertrain model with the only difference being the change of converter model and the addition of the ultracapacitor model. Also the change of brake model icon indicates the change in the implemented regenerative braking controller. It is important to note that the development method has not only allowed for a different power supply architecture to be implemented while reusing the existing architecture and models for the remainder of the powertrain model, but the

changes are also visually evident at the top hierarchical level of the model so that a new developer or user can easily identify differences between models. Having said this, it is of course still necessary for the developer to document the purpose of each saved model within the code description and link it to any related documentation as discussed in Section 7.1.

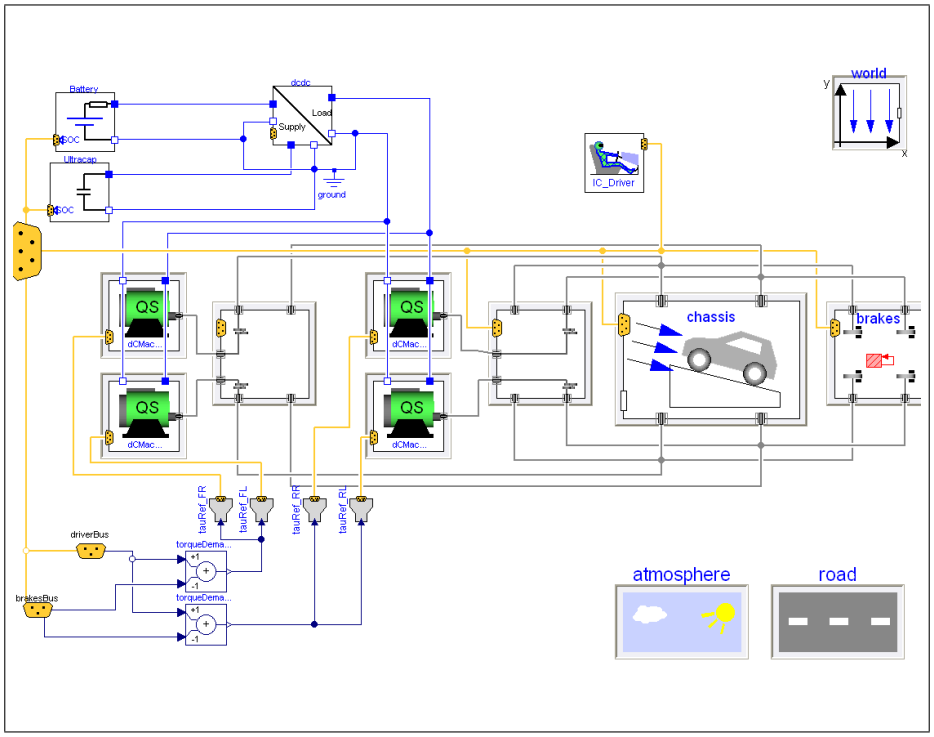



Figure 6.12: Powertrain model for LifeCar using battery and ultracapacitor power supply.

### 6.3.6 Dual Source Powered Model: EV Range Simulations

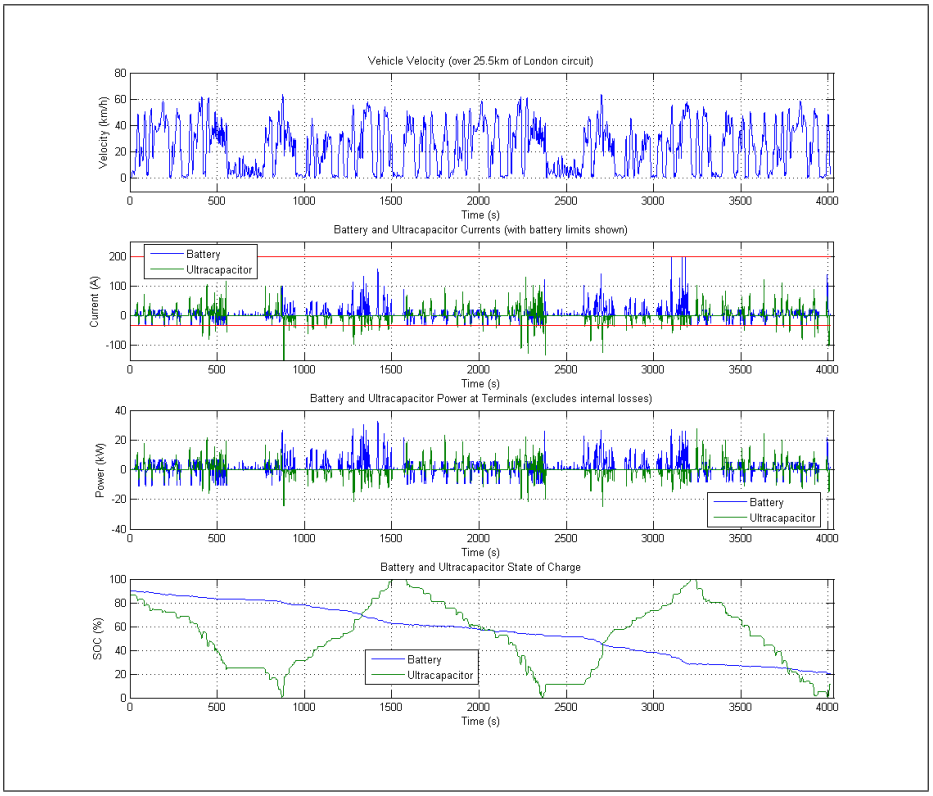
With the addition of the ultracapacitor to the power supply, the vehicle range increases from the 19.2 km for the battery powered option to 25.5 km. The main reasons for the original 30 km range that the power supply was designed for not being achieved, are the increased vehicle mass and the SOC limits implemented in the energy management strategy which are not optimized.

Figure 6.13(a) shows the current and power demands on the battery and ultracapacitor as well as their SOC over the 25.5 km distance of the London drive cycle. Here it can been seen that during the ultracapacitors discharge phase, all power demand

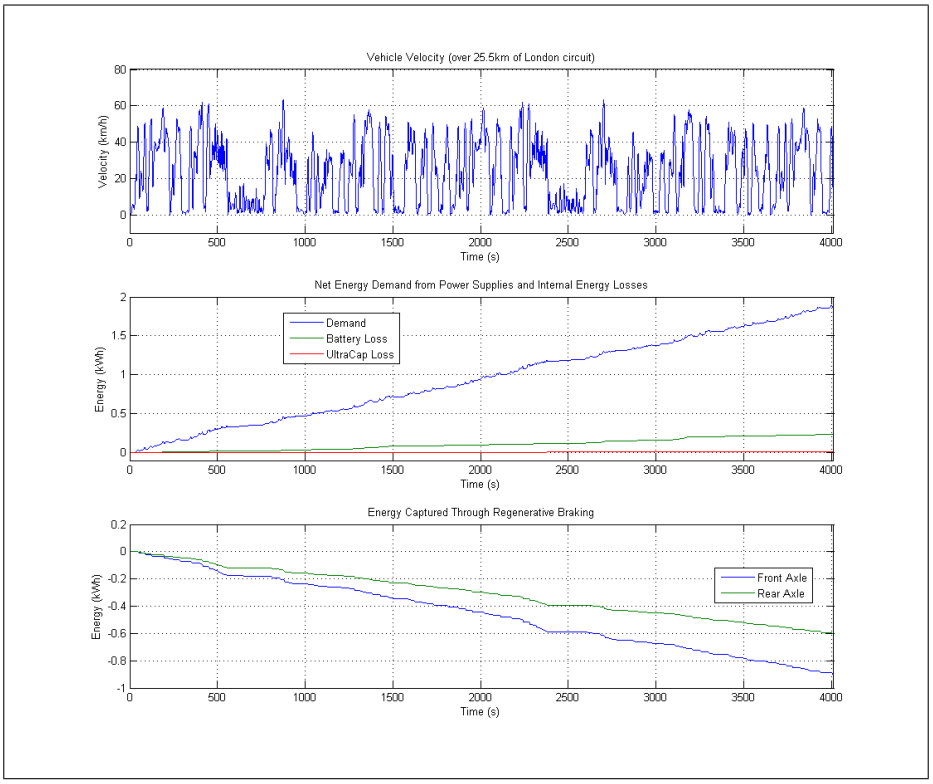
above 6.5 kW are met by the ultracapacitor while the bulk of the regenerative power is absorbed by the battery with only the excess being absorbed by the ultracapacitor. A base power limited by a battery current of up to 3C (24A) is always supplied by the battery, even during the ultracapacitor discharge cycle. This is done to help maintain a constant voltage since the $LiFePO_4$ batteries can maintain a constant voltage until a 30% SOC as long as the current drawn is below 5C [172] and increase the cycle life of the battery by lengthening the charge-discharge rate [165]. The author aknowledges that this strategy is not consistent witht the constant resistance battery model used but it was implemented with the idea of replacing the battery model with a map-based model using measured voltage discharge characteristics supplied by the battery supplier. Unfortunately this data was not obtained and the battery modelling is to form part of future work.

During the ultracapacitor charge phase, the bulk of the power demand is provided by the battery while all the regenerative braking power is absorbed by the ultracapacitor. Therefore, the battery SOC drops at almost double the rate it dropped during the ultracapacitor discharge phase. This effect is primarily due to the hysteretic charging and discharging cycles of the idealized energy management strategy. Ideally, a more precise controller would make use of the ultracapacitor for all peak power demands and take priority for charging during all regenerative braking events. The simulation ends when the battery SOC reaches the specified minimum value of 20%.

Figure 6.13(b) shows that the net energy demand for the covered distance is 1.88 kWh and the internal energy losses of the battery and ultracapacitor are 0.24 kWh and 0.01 kWh respectively, giving a total energy usage of 2.13 kWh. Finally, in the bottom plot of Figure 6.13(b), the energy recaptured at the front and rear axles due to regenerative braking is shown. With 0.89 kWh captured at the front of the vehicle and 0.59 kWh captured at the rear, the total amount of energy recaptured at all wheels during regenerative braking is almost double that of the battery powered model. These results indicate that hybridizing the power supply with the addition of an ultracapacitor has a definite benefit in recapturing braking energy and reducing the demands on the battery, thereby prolonging its duration and providing an associated increase in vehicle range.




(a) Current, Power and SOC




(b) Energy Usage

Figure 6.13: Simulation results for battery and ultracapacitor powered range test

### 6.3.7 Battery and Ultracapacitor Powered EV Performance Model

In the first case study, the high fidelity LifeCar model developed in Section 5.3.3 required new subsystems to be added to the HEV library according to the *specialization* step of the *Implementation* stage. In this case, since the specialized subsystems already exist in the developed subsystem library and no further specialization is required, considerable development time and effort can be saved by reusing these subsystem models. It is in this situation that we see the value of applying an object-oriented development method within a prototyping development methodology. With the development of each prototype, the object library keeps growing and development effort can focus on those objects that need improvement while carrying forward all useful objects from previous development cycles.

In particular, the performance model makes use of the same powertrain model as the EV range model in Figure 6.12, but with the chassis and tyre subsystems replaced with high fidelity models described in Section 5.3.3. Additionally, the vehicle mass is adjusted to account for the ultracapacitor bank using the same assumed mass distribution as for the battery powered EV model. This gives a vehicle mass of 519.7 kg on the rear axle and 480.3 kg on the front axle. Again, the data set [161] representing a dry road surface is used to parameterize the non-linear tyre models.

### 6.3.8 Dual Source Powered Model: EV Performance Simulations

With the addition of the high power density ultracapacitors to the power supply, the maximum acceleration achievable is increased with the vehicle being able to accelerate from 0-100 km/h in 10.6 s as shown in Figure 6.14(a). Examining the middle plot of Figure 6.14(a), it can be seen that the battery current increases to 200 A at which point the ultracapacitor supplies additional current to maintain the acceleration rate until it too reaches 200 A. The ultracapacitor current is limited by the 200 A limit of the power electronics in the converter.

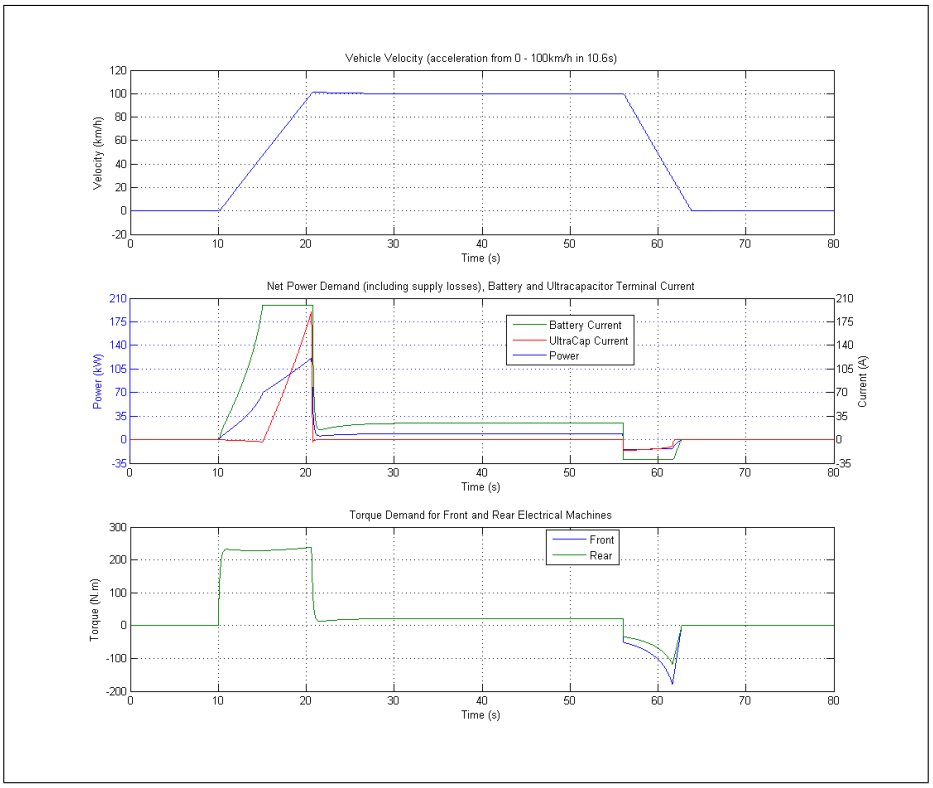
Since there is no dynamic control of the two power sources, the order in which the power supplies are used is dictated by the power balance implemented in the dual input converter model. The topic of managing and controlling the power split

in EVs and HEVs forms the focus of much research [67, 68, 174, 175, 176]. As pointed out in [177, 178], rule-based control approaches can be optimized in order to operate each power source as efficiently as possible, or static optimization can be performed for known drive cycles. However, these control approaches are only optimal for specific conditions and not any random driving pattern. Fuzzy and predictive control methods provide more flexibility in this regard but at the cost of increased design variables and computation time. A more realistic and practical control approach is to have a supervisory controller as proposed by Paganelli et al. [179] and implement different optimized power distributions based on the vehicle mode and the state of the power sources.

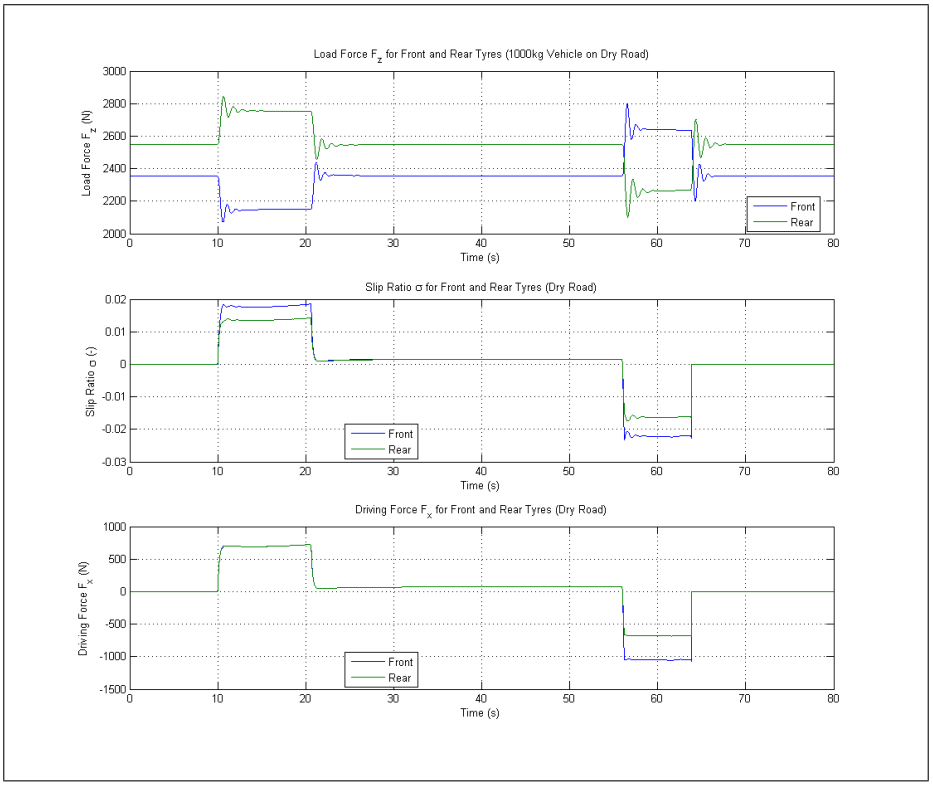
The net power demand during this acceleration peaks at 120 kW, which is almost double that achieved by the battery powered vehicle model. Even so, the bottom plot of Figure 6.14(a) shows that the torque demand on the electrical machines is only 240 N.m, which is less than half of the rated peak torque of the machines. In order to accelerate faster, a larger peak supply power is required, which can be achieved for short durations by allowing the ultracapacitor to discharge with higher currents. Since the ultracapacitor has a maximum peak current of 4000 A for short bursts of up to one second, a converter with a higher power rating is needed to allow currents above 200 A on the 400 V bus. If more power is required for longer durations, in addition to increasing the converter rating, a larger battery pack is required with cells connected in parallel in order to increase the current capacity of the battery pack.

Figure 6.14(b) shows that with a heavier vehicle and greater acceleration, with respect to the battery powered vehicle, there is a larger mass transfer to the rear and hence a larger wheel slip ratio at the front wheels. During braking the vehicle pitches forward placing more weight on the front wheels and together with 60% of the braking torque, this causes slightly less slip on the front wheels than the battery powered counterpart. The bottom plot shows that the vehicles braking force is split between the front and rear tyres according to the specified vehicle brake balance.

These results also indicate that the hybrid power supply is a better option than the battery alone. This is based on the fact that more power is delivered to the electrical machines, reducing the 0-100 km/h acceleration time by almost 60%. Further, with appropriate control, the ultracapacitor can be used to provide the peak power demands and thereby spare the battery and prolong its cycle life.




(a) Power, Current and Machine Torque Demands




(b) Load Force, Slip and Driving Force

Figure 6.14: Simulation results for battery and ultracapacitor powered performance test.

### 6.3.9 Front and Rear Wheel Drive Powertrains

The final set of executable models and simulations in this case study are those for the comparison of 4WD and 2WD LifeCar architectures. The main motivation for investigating the 2WD option is to determine where the requirements could be met by only two electrical machines given their high power and torque characteristics. Additionally removing two machines can simply the complexity of the vehicle control system requirements, reduce the number of auxiliary devices needed, reduce the production cost of the vehicle, provide more space on the chassis and decrease the overall vehicle mass.

More specifically, the powertrain mass reductions due to the removal of two electrical machines is explained with respect to the 4WD counterparts as follows:

1. Front wheel drive options

    - Two 25 kg electrical machines are removed from the rear of the vehicle. Therefore, the battery powered FWD vehicle is 50 kg lighter than the 4WD counterpart.
    - For the hybrid power supply option. Assuming that 60% of the braking force is provided by the front wheels, the ultracapacitor only needs to be sized for 60% of the uncaptured energy or 0.15 kWh. Sizing the ultracapacitor bank for this new energy requirement using equation (6.9), results in 81 cells of the 3000 F capacitors. This means that the new ultracapacitor bank weight is 14 kg lighter than that of the 4WD. In total the dual source powered FWD vehicle is 64 kg lighter than the 4WD counterpart.

2. Rear wheel drive options

    - Two 25 kg electrical machines removed from the front of the vehicle. Therefore, the battery powered RWD vehicle is 50 kg lighter than the 4WD counterpart.
    - For the hybrid power supply option. Assuming 40% of the braking occurs at the rear wheels, the ultracapacitor needs to be sized for 0.1 kWh. In this case only 69 of the 3000 F ultracapacitor cells are required, thereby reducing the 4WD ultracapacitor bank weight by 21 kg. In total the dual source powered RWD vehicle is 71 kg lighter than the 4WD counterpart.

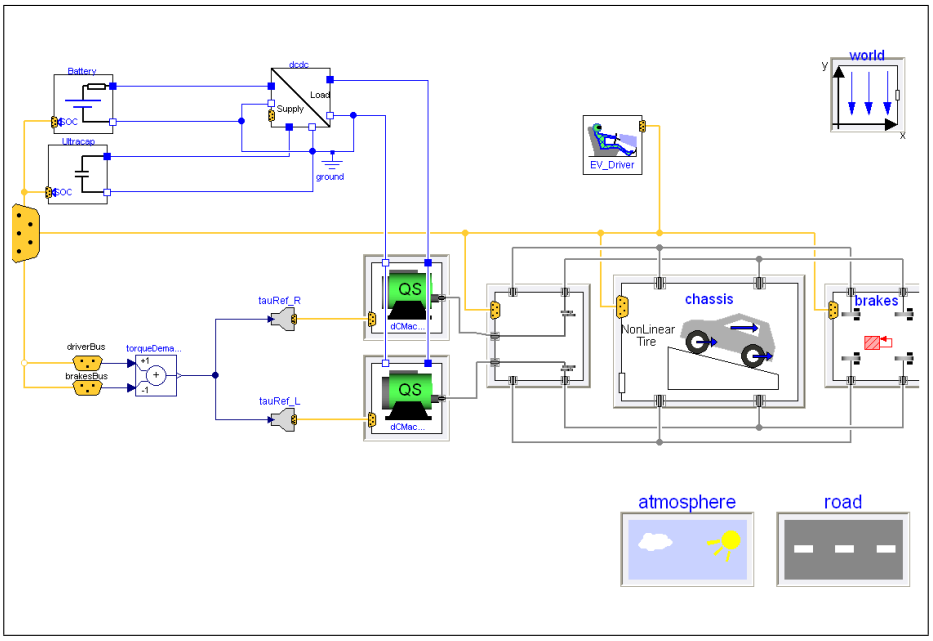



Figure 6.15: Example 2WD powertrain model for LifeCar using battery and ultra-capacitor power supply.

From a modelling perspective, this comparison exercise provides an example of how the modelling method is suitable for investigating design changes. In particular, the increased flexibility and reduced development effort gained by redefining the base model created in Chapter 5 is seen. Since the 2WD base model shown in Figure 6.6 has been added to the HEV model library for this case study, developing the four executable models required to investigate the 2WD vehicle architectures is greatly simplified.

Specifically, all required subsystem models are now available in the HEV library, and therefore creating the new architectures is a matter of inheriting the base model and instantiating the desired power supply subsystems within the model. Additionally, due to the decision to implement the driveline models as two input models as discussed in Section 5.2.2, changing between FWD and RWD options is done by redeclaring the driveline model from a front wheel to rear wheel driveline. Once the complete vehicle model is defined, this same model is reused in the performance simulations by redeclaring the chassis subsystem. Figure 6.15 shows an example of a RWD drive powertrain with the hybrid power supply and the high fidelity chassis subsystem for EV performance simulations.

**EV Range Comparison**

A comparison of the maximum range achievable and energy recaptured through regenerative braking over this range for each of the powertrain architectures and power supply topologies is shown in Table 6.9. The vehicle range is determined by repetitive cycling of the London drive cycle until the power supply becomes power limited or the battery reaches the specified minimum SOC.

Table 6.9: Vehicle range comparison of 2WD and 4WD powertrains.

| | Power Supply | Mass [kg] | Range [km] | Captured Braking Energy [kWh] |
|---|---|---|---|---|
| RWD | Battery only | 890 | 17.1 | 0.25 |
| RWD | Hybrid | 929 | 18 | 0.4 |
| FWD | Battery only | 890 | 18.4 | 0.42 |
| FWD | Hybrid | 936 | 20.36 | 0.7 |
| 4WD | Battery only | 940 | 19.2 | 0.77 |
| 4WD | Hybrid | 1000 | 25.5 | 1.51 |

Comparing the RWD and FWD architectures, it can been seen from this data that the FWD powertrain recaptures more braking energy and provides a marginally longer vehicle range. This is the case for both the battery and hybrid power supply topologies, with the FWD battery powered architecture capturing 68% more braking energy than the RWD, and the FWD hybrid powered architecture capturing 75% more. These differences can be directly attributed to the longitudinal brake balance and the fact that there is more kinetic energy in the front wheels due to the mass transfer of the vehicle during braking. Also, when comparing the hybrid supply with the battery supply, though the captured energy increases by over 60%, the vehicle range increases by 2 km at best. This would indicate that hybridizing the power supply does not provide much benefit in terms of extending the vehicle range.

With the 4WD architecture, the ability to recapture braking energy at all four wheels does not provide much benefit when comparing the battery powered 4WD results with the hybrid powered FWD results. In this case, the better option is the hybrid FWD since it can recapture almost the same amount of energy but has a longer range due to the vehicle being slightly lighter. However, when the hybrid power

supply is applied to the 4WD architecture, this results in the heaviest vehicle and an increase in captured braking energy of 96% which in turn extends the vehicle range by 33% or 6.3 km.

### EV Performance Comparison

A comparison of the acceleration performance of the 2WD and 4WD powertrains for both power supply topologies is presented in Table 6.10. Here it can be seen that for the battery powered vehicles, despite the 50 kg reduction in mass, both the FWD and RWD are able to accelerate from 0-100 km/h less than 2s faster than the 4WD counterpart. This is because the main limiting factor is still the maximum power that can be supplied by the battery. Also, the RWD powertrain is slightly faster than the FWD due to more mass and increased tractive force on the rear wheels during acceleration.

For the hybrid supply powered vehicles, the mass advantage is smaller and the decreased ultracapacitor size is a disadvantage in terms of available power. Therefore, the 4WD vehicle has the fastest acceleration due to the higher power availability. In the case, the FWD vehicle is faster than the RWD counterpart despite the mass transfer to the rear during acceleration. This is because the ultracapacitor of the FWD powertrain was sized for 50% more energy storage than the RWD powertrain, which resulted in the 17% more power availability from the ultracapacitor.

Table 6.10: Vehicle acceleration time comparison of 2WD and 4WD powertrains.

| | Power Supply | Time (0-100 km/h) [s] |
|---|---|---|
| RWD | Battery only | 24.55 |
| RWD | Hybrid | 13.58 |
| FWD | Battery only | 24.8 |
| FWD | Hybrid | 12.8 |
| 4WD | Battery only | 26 |
| 4WD | Hybrid | 10.6 |

## 6.4 Discussion

During the course of this case study, several lessons were learnt with regards to the modelling and simulation process. These are listed below where the first two points reinforce the observations made in the Chapter 5 and the last two points refer to more general requirements of the modelling environment.

1. Flexibility and Modularity

   The ability to create interchangeable models with varying degrees of fidelity is key in providing design flexibility. Using an object oriented design approach ensures that a design will be modular as far as the subsystems and components are concerned.

2. Generic Models

   Once again it was shown that it is difficult to make a decision as to where in the hierarchical abstraction the design should make a transition from generic to architecture specific. Having a good understanding of the scope of the design and possible future developments can help in making this decision. Trade-offs remain between the available design flexibility, the design effort and the maintenance effort required.

3. Linked Database

   In order to simplify the parameterization process, especially when iterative studies need to be undertaken, with various parameter sweeps being performed or any different parameterization of the same underlying model, it is possible to make use of data records. A data record is an instance of a generic database containing all required parameters for a model. The model is designed to point to this database for all its parameter values and different parameterizations can be achieved by simply changing the data record and not having to save a new instance of the model. CAE applications can then be implemented to automate the creation of data records, either from a central data repository or directly from other software packages where the required data is known. This helps both in minimizing the risk of human error when entering parameter data into a model, and in moving towards a more automated development environment as mentioned in Section 7.2.2.

4. Debugging

   The issue of debugging is directly related to the modelling environment being used. In this case the equation-based modelling language makes debugging more challenging since the language is designed to allow a high-level of abstraction. The higher abstraction level means that the abstraction between compiled simulation code and the original model code is much greater and hence it is more difficult for a compilation error to be traced back to its original source in the model.

## 6.5 Conclusion

Referring back to Chapter 1, the objective of this research is to provide a development method that facilitates the exploration of the HEV powertrain design space. In other words, the development method must offer the developers sufficient flexibility to pursue many ideas in as fast and efficient a manner as possible. This case study established the flexibility and suitability of the proposed method for exploring multiple design options in a timely fashion. In this case the proposed method was applied at the initial design stages of a real world HEV development project. The method was employed for some concept evaluation and analysis with regard to the vehicle topology and power supply architecture. Once again, the hierarchical nature of the model library proved to be ideal for reducing model development time, while the object-oriented properties lent themselves well to the extension and maintenance of the developed model library. Using replaceable subsystem models allowed for the efficient evaluation of several designs with minimal redundant modelling effort.

From a systems engineering viewpoint, this case study mimicked a prototyping or spiral type life cycle where several design options are evaluated and improvement is made in an incremental manner. The iterative nature of the development method provides vital feedback to the systems processes for incremental design. Specifically, by using OOM for concept analysis and evaluation through the development and testing of executable subsystem models.

Further, analysis and feedback with regards to the sizing of the power supply in order to meet the performance requirements, contributed to the refinement of requirements and the selection of an initial battery pack solution for the Morgan LifeCar2 project.

# Chapter 7

# Model Management

In order for any modelling and development project to be carried out successfully, it is imperative that there be some form of appropriate management structure in place. Furthermore, the management process is even more important when a major change in development methods is being implemented. For example, Fayad et al. [180] show that the management processes required for making the transition to object-oriented software engineering had to be specifically designed for the object-oriented paradigm. In particular the management activities included how object development is tracked, evaluated, documented and maintained.

The field of software, systems and project management is vast with many methods, tools and techniques being used in industry today. It is beyond the scope of this Thesis to cover all the facets of managing a HEV modelling and development project. Instead, this chapter will focus on those elements of management deemed most relevant to the implementation of an OOM method with respect to the issues that have been highlighted in the preceding chapters. The chapter first considers features regarded as necessary at the modelling environment level, and then considers some of the management aspects at the system integration level.

## 7.1 Modelling Level Management Concepts

### 7.1.1 Maintenance and Modification

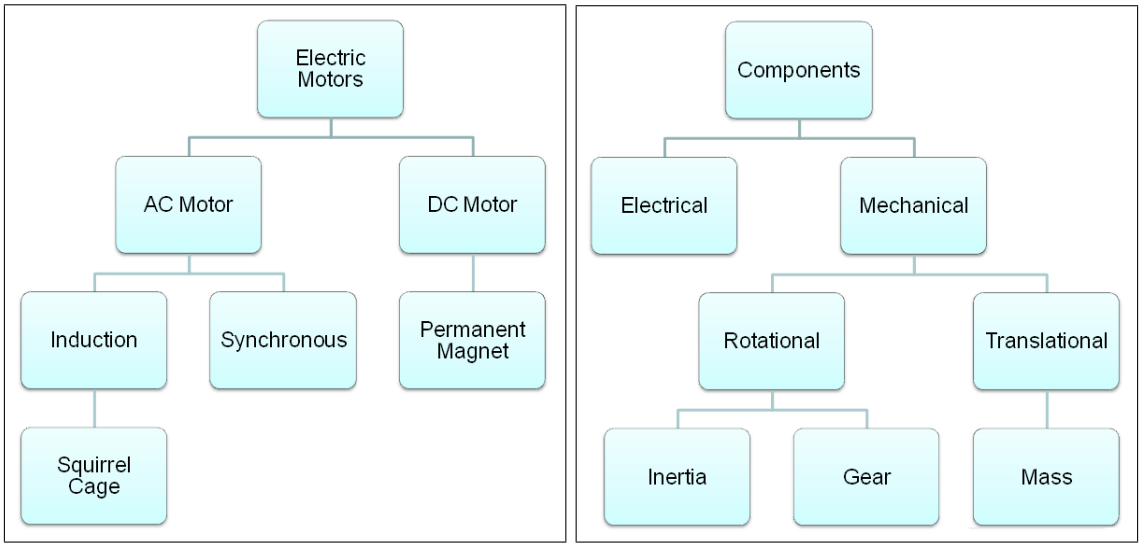
In Chapter 1 it was stated that the increased complexity of modern vehicle designs, and in particular HEV designs, leads to increased model maintenance and management issues. Further, object-oriented modelling was proposed as a possible solution to this problem. Features of object-oriented design such as inheritance and encapsulation may promote code reuse and simplified maintenance. However, the reusability and efficiency of the models does not come from the use of object-oriented tools or languages, but rather from how these features are implemented in the development process and how well this process is managed [100, 180]. Jennings and Rangan [5] claim that a pilot model management system implemented at Ford proved that the management of models for reuse required increased effort and commitment from developers as opposed to the management of stand-alone models.

#### Model Library Structuring

Fichman and Kemerer [100] point out the need for "harvesting" reuse by finding and exploiting existing knowledge, components and models. The main mechanism for sharing previous design knowledge and effort between different designs and developers is the use of a model library and corresponding data repository. At the modelling level a model library is required to establish abstractions for different domains and to provide the foundation for reusability, modularity and collaboration by different users [5]. Further, the use of model libraries reduces the time spent on testing and validation by allowing for the reuse of well tested models [76].

In this Thesis the model library structure is implemented as a Dymola library with the abstractions for models in each domain being represented by partial models with domain specific connectors. As was pointed out in Section 4.4.3, there are different approaches that can be taken to create the model library structure. Two of the more common and intuitive ways are to create a hierarchical structure classified by either function or domain.

A more generic function based grouping on upper levels of the hierarchy, with models becoming more specific on the lower levels. This is illustrated in Figure 7.1(a) with




(a) Hierarchical library classified by function (b) Hierarchical library classified by domain

Figure 7.1: Example model library structures

the functional hierarchy of electric motors. At the upper level a generic or base model for electric motors can be stored with lower levels inheriting the upper level class and properties. In this way the code reuse is enforced and maintenance is simplified. Further, when adding a new specific component to any level of the hierarchy, the developer can concentrate on only implementing new features that are not present in any of the upper level models. In other words only the specialization needs to be implemented since all features of the upper levels will be inherited.

A domain hierarchy is better suited for bottom-up construction since there are many basic component models which will be used for the creation of more complex system models. This method is more intuitive for initial development efforts as component models can be can be searched for in the same manner as they would in a real-world design. An example of a domain based library structure is shown in Figure 7.1(b). This structure is adopted by the Dymola tool in order to provide the basic component models supplied by Modelica standard library.

The functional classification approach to library structuring is taken in [13] where the inheritance mechanism in the structure is used to ensure that descendant component models can replace their more generic ancestor models, thereby allowing the model to evolve throughout the design life-cycle. Han et al. [13] also stress that one component model can form part of different functional hierarchies if it can be classified in different ways. For example a rotary motor can be seen as part of a transducer

hierarchy and a mechanical hierarchy if it is considered as a rotational joint. This has the added benefit of allowing future developers to find the model for reuse in different application domains while still only having one model to maintain.

It is possible to make use of both of these library structures by having more than one library. Eriksson and Jacobson [63] follow this approach by making use of a domain based component library and a second system library where models are structured hierarchically according to a specified system decomposition of a vehicle. A similar method was employed for the case studies in Chapters 5 and 6 since the basic components used were mainly sourced from external libraries while the developed subsystem models were packaged according to the identified HEV subsystems.

Overall the objectives for a library design should be to promote object-oriented modelling by providing developers with a structure within which object-oriented properties, such as inheritance and encapsulation, can easily be implemented by following the library structure. Further, users of the library are provided with an easy and intuitive way to locate models for use in new modelling activities. As with the design methodologies, there is not one approach that will provide the best results for all development projects. However, it can be said that whatever library structuring approach is decided upon, it is imperative that the management structure enforces its adoption and does not allow developers to use an ad-hoc structure. Some guidelines that can be used in the development of a library structure are listed below [13, 136, 181]:

- promote decomposition and composition of subsystems and components,
- reflect model architecture in the library structure,
- make a clear distinction between generic and specific models,
- provide a means of implementing different levels of fidelity,
- facilitate locating components and subsystem models for new designs,
- define a means for making models replaceable,
- provide a basis for collaboration between domain specialists and other members of the supply chain.

Management of the library structure is not only needed at the development level, but the model library should be closely tied to a data repository at the system level of operation. This data repository should link all related documentation such as CAD data, requirements specifications, design decisions and parameter data so as to form a unifying collaborative base for capturing, sharing and reusing knowledge from all stakeholders.

### Version Control

Due to the multidisciplinary nature of HEV powertrain development, engineers from different domains and with different specialities are required to develop the complex model of the powertrain and its subsystems. Model developers range from engine and battery specialists to electronics and control specialists, with each party making their contribution to the powertrain model at different times. For this reason it is imperative that there is a process of version control and management in place that can assure that all parties are using the most recent model and that all contributions are added to the same model.

Further, with development activities occurring in parallel and possibly in different locations, it is necessary to have a system in place that ensures the correct model data is stored and shared for all developers [60]. A version management system should be used to keep track of model code, documentation, parameter files, test scripts and simulation results. The task of version management is usually handled by bespoke software if not built into the development environment. Some of the basic features that should be supported are:

- Add models to a repository. This feature is necessary to support the extension and evolution of the model library by making newly developed component and subsystem models with associated documentation available to all collaborating parties.
- Updating local files. This feature is useful when files or models have been stored off-line for checking if there are differences from the repository version and exactly what those differences are.
- Commit changes to repository. Once an developer has performed a maintenance task on a model or has upgraded the model, those changes can be up-

dated in the repository by comparing with the repository version and providing comments as to what changes were made. These comments are necessary for traceability over the design life-cycle.

- Merging different versions (with conflict handling). In a multidisciplinary collaborative design environment, it is not desirable to prevent access to models by locking all files being used [60]. Since it is likely that more than one developer will need access to the same model, especially in an object-oriented modelling environment, the system should allow for concurrent development. Therefore, the ability to merge the concurrent changes made to a single model is neccesary. If two developers have made changes to the same lines of code or documentation, the system should issue a warning and allow the user to resolve the problem.

- Revert to previous version. Modelling and analysis activities must not be halted. If a version update to a particular subsystem model creates errors in the working model, reverting to a previous working version allows for modelling and integration activities to continue, while debugging of the version update can occur in parallel.

### 7.1.2 Documentation

As mentioned before, HEV modelling is a multi-disciplinary task that requires the involvement of many stakeholders. Communication between these stakeholders is key in order to ensure project success. Much of this communication is done in the form of documentation which is passed around via email or shared in a common workspace and may have the form of spreadsheets, text documents, diagrams or presentations. Managing the documentation that accompanies each developed model is important in order to track the progress of a project and trace any changes that may have occurred during the development life-cycle. Often the documentation is imperative for maintenance tasks as it is the only available link to the original design [180]. Lohmann and Marquardt [182] explain that 50% of modelling time is spent looking for known information and that this process can be supported with improved documentation. This documentation should support model reuse by allowing users to easily understand how a model can be used without going through the underlying code. Model reusability is also aided by documenting experience

gained such as solutions to particular design problems.

More specifically a documentation standard should be enforced for documents produced during the modelling activities. This standard should ensure that developers correctly document their developed models by:

- stating the requirements for the model,
- stating the development rationale employed,
- tracking any decisions made with regards to implementation or maintenance,
- clearly stating any model dependencies.

If an overall system model is created using the UML or the SysML, the diagrams developed to describe the system model will form a major part of the documentation containing knowledge and contributions from all stakeholders. Configuration management can be employed in order to define what diagrams and documents need to be created and controlled during the design life-cycle. Further, since these configuration items are all constantly being modified during the development process, they should all be added to the version control repository [180].

### 7.1.3 Model Sharing and Protection

Another factor that needs to be taken into account when dealing with large complex projects such as those associated with HEV development, having multiple stakeholders that are possibly distributed in different geographical locations, is the protection of intellectual property (IP). Tummescheit [76] points out the fact that in the commercial software industry, a line of validated code is given a value ranging from $50 to $200. This number is possibly even greater if the code is specialized modelling code. Therefore, as the technology being investigated becomes more mainstream, the value of the tested and validated models will increase. As these technologies are taken up in industry, it is reasonable to assume that the parties who have invested time, effort and money in developing the model library resources will seek to protect there IP by encrypting or hiding proprietary sections of their model code.

The Dymola User's Manual [60] explains that a common method of sharing models or software without revealing the underlying source code is to provide users with

compiled executable programs. It also points out that this method is not suitable if the intention is to use that model within an equation oriented modelling environment. This is explained as follows [60]:

> "This approach is not useful for Modelica models. To achieve robust and efficient simulation, it is important that Dymola can make a global analysis and manipulation of all equations. It is thus highly desirable to give Dymola access to the equations in their original form. Encryption of the textual Modelica representation of the model supports concealment of internal parts such as the equations, while still allowing Dymola internally to access the equations as if the model was not encrypted."

In practice this means that an encrypted model is made available as a black box model meaning that the user is able to see connections, parameters and documentation but cannot see or change the internal equations describing the model's behaviour. This is put into effect by the use of `public` and `protected` sections within the Modelica code description. The implication of developing encrypted models and libraries is an increased development and testing effort since care must be taken to provide potential users with sufficient access to parameters during modelling and simulation, while separating those parameters that should be concealed. Further, since the user can not make modifications to the underlying code, more extensive testing of the model should be performed since any errors experienced by users will be relayed back to the developers for debugging increasing the maintenance costs.

The sharing and encryption features discussed so far are restricted to the use of models within the same modelling and simulation environment, or at least one that is able to make use of native Modelica models. However, with complex multi-domain development projects, it is often necessary to be able to share models between different modelling and simulation environments. This ability is needed to allow for further development of other aspects of the design without the need to recreate existing models in a different language. An example of this was seen in Chapter 5 when the Dymola vehicle plant model was exported to the Simulink modelling environment in order to test the plant with controller models developed in Simulink. Since Simulink is the industry standard for controller development, most modelling and simulation environments support the exporting of models into S-functions that are suitable for use in the Simulink environment. A survey of vehicle simulation

tools used in industry performed by Ricardo in August 2009 shows that 90% of the surveyed tools support the generation of S-functions [183].

## 7.2 System Integration Management Concepts

Colombi and Cobb [116] highlight how well managed systems engineering can effectively be used for development projects in a critical environment, making the following key observations:

- Early statement of key constraints helped in focusing all stakeholders, making project decisions and conducting trade-offs.
- Understanding the larger system context helped tailor the development approach.
- Software engineering techniques could be used for rapid hardware development, such as evolutionary development and rapid feedback.
- Selection of a suitable toolset was important in aiding the decision-making process.
- Project risk could be reduced through parallel development.
- Responsive development can be achieved while maintaining the rigour of classic systems engineering methodologies.

This section considers model management from a higher level by considering how the modelling process fits into the overall system engineering process. Particular emphasis is placed on the following three areas:

1. Managing stakeholder communication – stakeholder involvement is a key aspect in HEV design due to the multidisciplinary expertise required to support the development life-cycle.
2. Computer aided engineering – considering the high number of tools already being used within the automotive industry, introduction of new tools and methods for HEV design needs to be well managed in order to move towards a more streamlined and automated development environment.

3. Risk management – since the proposed modelling method is to be used within the framework of a systems engineering development methodology, risk analysis should form part of the development life-cycle. In particular, managing the risks associated with implementing new technologies and changes to procedures.

### 7.2.1 Stakeholder Communication

When dealing with multiple stakeholders each needing to make a vital contribution to the requirements of the project, it is crucial that the communication between the stakeholders be well managed. McManus and Wood-Harper [131] analyse the causes 42 information system project failures and shown that 65% of the project failures can be attributed to management issues, with poor stakeholder communication and risk management being significant causal factors. Two specific aspects of the stakeholder communication that need to be managed are the requirements capture and data exchange mechanisms.

Stakeholder requirements need to be captured, analysed, shared, tracked, updated and validated. Bahill and Henderson [184] stress the importance of requirements development, verification and validation by offering examples of famous projects where neglecting one or more of these steps led to project failure. Examples include the HMS Titanic where poor quality control meant the verification process was not performed correctly and the Mars climate orbiter where different stakeholders made use of different measurement units causing the satellite to crash.

Capturing and modelling the requirements is a key role in designing HEV powertrains [23]. Making use of an overall system model in the form of a SysML model is a useful tool for communicating and capturing different stakeholder requirements as well as sharing the captured data when used together with a version control system. The SysML provides for requirements modelling through the use of requirements diagrams that can be integrated for use with requirements management tools [185]. Since the requirements and associated documents will change as stakeholders interact and the design evolves, the requirements management tool should provide a database interface for capturing and tracing the changing requirements [186]. Additionally such a tool will help in preventing redundant document creation and thereby promote data consistency.

### 7.2.2 Computer Aided Engineering

In an effort to better support co-simulation between different modelling environments, the latest release of Dymola (version 7.4) includes support for a model exchange standard known as the functional mock-up interface (FMI). The FMI is an initiative of the MODELISAR project of the Information Technology for Information Advancement, or ITEA2, program [187] that defines a common interface for exchanging models between different tools. The main objectives of MODELISAR are to allow for the concurrent design of embedded software and systems by combining the Modelica and AUTOSAR standards and to ensure runtime interoperability between different tools for co-simulation through the FMI [143]. Essentially this means that the different modelling tools supporting the FMI interface can exchange models in the form of a unit consisting of a standard model description, a source code implementation model in C, input parameter resources and the documentation for the model. As of 2010, several manufactures of modelling and simulation software intend to incorporate support for the FMI into their simulation and modelling environments. This includes multi-domain simulation packages such as AMESim and SimulationX, integration and co-simulation software such Silver 2.0 and EXCITE ACE and 3D multibody simulation such as SIMPACK [84, 143].

In an interview with a technical specialist in the simulation group at Jaguar Cars Ltd., it was revealed that the company had over 100 different computer aided engineering (CAE) tools in use for different modelling purposes such as CFD, combustion analysis and vehicle dynamics [129]. Further, the main method for data exchange was via spreadsheets and presentations, and data consistency managed during meetings with the different stakeholders.

In practice, well managed sharing of models and data can save time and effort by avoiding duplication, and simplifies the task of checking the consistency of the data. More specifically it is a necessity if the objective is to move towards a more automated environment. Automating and controlling the exchange of data removes many of the opportunities for errors to be introduced into the process, typically caused by users manually transferring data from text based communications.

### 7.2.3 Risk Management

When managing the development of a technological system, there is a risk of project failure that can be caused by the management, the users, the hardware or the software [188]. In the software domain risk is used to indicate the possibility of factors such as delays, rising costs, low quality solutions or project failure [189]. Risk management needs to be integrated into the design development methodology and occur continually throughout the development life-cycle. During the first stage of the modelling method, *Functional Requirements*, risk is reduced by defining realistic requirements which will form the guidelines for later development efforts to focus in the correct areas. Typically, risk is high at the start of the project when there are more unknown variables and it is reduced through analysis, experimentation and testing as the development advances [190].

As has been stated throughout this Thesis, HEV's are highly complex systems and their modelling and development requires the participation and collaboration of multiple stakeholders. Further, exploring the HEV design space involves investigating new and emerging technologies in order to gain a competitive advantage. From a software and systems point of view, it can be said that the risk likelihood or probability of failure is high when dealing with new and unproven technologies, or highly complex components [190]. This is because such design efforts must significantly extend previous designs while working with minimal data availability, complex or newly developed components and a general lack of data and practical experience [190].

Though there are many risks that can be associated with the running of any project, it is beyond the scope of this research to highlight all the risks involved with developing new products in the automotive industry. Rather, five risk aspects have been chosen which directly relate to the implementation of an object-oriented model development method such as the one proposed in this Thesis. The first four areas of potential risk are related to model development and are discussed with reference to the mechanisms that should lead to the reduction or elimination of that risk.

#### 1. Modelling Errors

These are errors that could manifest at a subsystem level or once the overall system model is put together. From a development point of view it can be said that

these risks can be mitigated by means of a good development method [191]. Specifically, features of the development method such as abstraction, the model library and simulation testing are designed to reduce the risk of model errors by reducing complexity, reusing tested components and validating functionality. Further, employing a development methodology such as the one shown in Figure 4.2, provides a means for the early identification of errors through the frequent iteration of the model development.

From a risk management perspective modelling errors can be avoided by enforcing technical reviews of developed model classes along with there associated documentation to ensure that models are corrected before being added to the model library for reuse in other models. Further models that are particularly problematic to implement should be tracked and reviewed more frequently in order to find a solution. Also any problems specific to a particular model need to be documented and stored so that the same problem can be avoided in the future.

**2. Model Fidelity**

The level of modelling fidelity is a risk factor both from an under performance and an over performance point of view. An inadequate model fidelity will fail to meet the modelling requirements while too high a fidelity could lead to an overrun on development time. The first way this risk is reduced is through the requirements specification and the iterative feedback from the development stage simulations to check if requirements are met or being changed. During the object-oriented modelling process, the properties of encapsulation (replaceable models) guarantee the implementation of a specific fidelity level does not affect external subsystem models and inheritance reduces the risk by allowing the incremental development of higher fidelity levels.

The main management action that needs to be carried out in order to mitigate these risks are to provide sufficient opportunity and mechanism for stakeholder communication so that requirements and data are up to date and consistent. Additionally it may be necessary to consider the overall system and adjust the developer's requirements if the solution is not practical from a cost and complexity standpoint. Also when a new technology is being introduced to the vehicle system such as a new type of battery or fuel cell, it can be difficult for the correct model fidelity to

be achieved either due to lack of in-house expertise or lack of available data on the technology. In this case a management decision to acquire the services and expertise of an external consultant may be necessary in order to reduce the risk of project failure.

### 3. Design Changes

Design changes should generally occur either because of a requirements change due to stakeholder needs, or technology changes, or a major model maintenance event. The primary process for dealing with this risk is through an iterative development process, where the use of prototype model designs can help inform early decisions on possible design changes. Following this, the use of partial models together with the model library means that compatibility of new model subsystems can be ensured while minimizing design effort and error through reusable component models. Further, through the property of encapsulation the affect of subsystem design changes on functionality is reduced, and can be more easily propagated throughout the whole model.

Once again, timely communication between all stakeholders is the key management task for avoiding this risk factor. Further, to avoid the risk associated with implementing a new technology into the vehicle design, that has not been successfully implemented before, it may be prudent to manage an alternative option. This alternative can serve as a backup in the event that the new subsystem's implementation fails to meet the anticipated objectives. A similar approach is used in software development, when the perceived risk of using a new design technology is at the highest level. Using a second design approach, at the risk of increasing cost and reducing performance, is a preferable alternative to that of a non functional system. In this way the success of the overall project does not depend on that of a single subsystem.

### 4. Third Party Models

The use of third party models can be due to the availability of commercial model libraries or external parties developing some of the required models. Possible shortfalls in using these models are due to poor functionality, inadequate documentation

or model accessibility issues due to encryption. As in the case of model fidelity risk, encapsulation and replaceable models help in shielding the overall modelling activity from problems with third party models and allow for alternate models to be easily substituted if necessary.

For this scenario the management policies that control the documentation are important both for keeping the external documentation with the third party models and for link in-house documentation on experience with using these models so that this knowledge can be shared with all users. If the problems cannot be resolved internally, a management action to have the third party developers or external knowledge source consult on the correct usage of the models is required. Failing this, an alternate package should be acquired or a substitute model developed internally.

These four development risks and the recommended mitigation features of both the development method and the management decisions are summarized in Table 7.1.

Table 7.1: Model development risk factors and mitigation features.

| **Risk Factor** | **Development Feature** | **Management Actions** |
|---|---|---|
| Modelling error | Abstraction | Technical Reviews |
| | Model Library | Tracking |
| | Simulation Tests | Documentation |
| Model fidelity | Requirements | Stakeholder Comms. |
| | Encapsulation | Requirements Adjustment |
| | Inheritance | External Consultant |
| | Replaceable Models | |
| Design changes | Iterative Development | Stakeholder Comms. |
| | Partial Models | Alternative Option |
| | Model Library | |
| Third parties | Encapsulation | Documentation |
| | Replaceable Models | External Consultant |
| | | Alternative Option |

### 5. Changing Methods and Tools

The fifth risk area is related to the organizational management structure with regards to how the change in model development methods and tools is implemented within the organization. Building confidence in new procedures and tools requires good stakeholder communication, particularly between developers and management. Kwak and Stoddard [191] explain that disjointed relations between developers and management leads to misunderstanding and trust issues. Khalifa and Verner [192] study the driving factors for 82 software developers that lead them to choose between a waterfall development method or prototyping method. In this study they determine that the two main drivers for adopting a particular method are: the "facilitating conditions" within the organization, such as the companies adoption of new technologies; and the "perceived consequence" on the development process quality rather than the product quality. This latter fact is attributed to the belief that "a quality process will result in a quality product".

However, in a later study, Riemenschneider et al. [193] show that a primary driver for developers is whether or not the methodology will improve their productivity and perceived performance, due to organizational reward structures. They further explain that the benefit of the methodology to the organization will not encourage its adoption even if it is mandated. Therefore, in order to effectively introduce new development environments and methods, the company must either demonstrate the potential for improved individual performance or explain how individuals will be rewarded for improvements in company performance.

Riemenschneider et al. also explain that developers will resist adopting methods that are not compatible with their current work processes. Management should avoid radical changes, clearly illustrate similarities between the development methods and possibly introduce the new methods in an incremental fashion. One manner of achieving this is to arrange pilot development programs within the company in order to test new tools and methods. Finally, it is important that the new methods are actively promoted by management in order to reduce the risk of them not being adopted due to peer developer's disapproval of the methods [193, 194].

# Chapter 8

# Conclusions

Traditionally multiple models of the same plant have to be developed and maintained in order to perform the required simulation analysis during the vehicle design life-cycle. Considering the increased time pressures, design alternatives, changes in technologies and overall complexity of the modern vehicle design space; continued usage of this approach creates more opportunities for the introduction of errors into the design process. Further, with increased system complexity, development and management of the many required models becomes more difficult and time consuming. This PhD Thesis considers the problem of the increasing complexity of modern automotive powertrain modelling and the fact that this problem is compounded within the HEV design space. A modelling method using object-oriented modelling principles within a systems engineering development environment was proposed as a means of dealing with this complexity, from both a development and management perspective.

The work presented in this Thesis focused on the intersection of three research areas. The first area being that of HEV powertrain modelling. Within this area it was established that the two main facets giving rise to the complexities in HEV powertrain modelling are the cross-coupling of multiple engineering domains and the varying levels of fidelity needed for different modelling objectives. Where the first facet adds to the architectural diversity of the HEV design space, while the second adds to the required logical complexity.

The second research area is that of OOM. Here, the concept of an equation oriented object-oriented modelling language was introduced with the advantages of acausal

component based modelling and visually intuitive model topologies, over traditional block based modelling techniques. Additionally, benefits are gained from the structural and behavioural properties provided object-oriented languages. In particular the use of inheritance within model development allows for reduced modelling effort due to reuse of code, and simpler model maintenance since only the original model class has to be maintained and not the individual instances.

The third research area considers the concepts of systems engineering methodologies as applied to the intersection of the first two areas. More specifically the concepts of systems modelling and development are applied to the HEV modelling domain in order to produce an OOM method. The proposed modelling method aims to provide a flexible manner of developing HEV powertrain models using object-oriented principles to meet the complexity challenges, as well as providing a mechanism for structuring model libraries so as to assist in the management, development and maintenance tasks. Further, it is intended that this modelling method should be easily incorporated into and take advantage of the already existing systems engineering practices within the automotive industry.

In Chapter 1 Section 1.3 an objective is set out for developing a modelling method that can take advantages of OOM and can be integrated within a systems engineering framework. This objective is met in Chapter 4, where a proposed a solution for the efficient exploration of the HEV design space is placed within the context of a systems engineering life cycle, and then explicitly defined as an OOM method for developing HEV powertrain models. Further, the proposed method is tested and the aims set to validate the objective in Section 1.3 are specifically met in Chapters 5 and 6.

In Case Study 1: LifeCar, the modelling method is used to develop a library for modelling the LifeCar fuel cell hybrid vehicle powertrain. This study showed how partial models can be inherited to more efficiently develop and maintain a variety of complete models. With this regards it is also highlighted that a balance must be achieved between the desired reusability and extendability of these models. Where a partial model needing more extension to become functional is likely to be reusable in more instances than one with more complexity, while the more complex partial model saves can reduce the development effort in terms of the quantity of reusable code. This case study also provided insight into the development and structuring of a hierarchical model library. The library employed a high-level package structure

based on function, with partial models and base models grouped by subsystem and each subsystem package containing subpackages for the different levels of fidelity. Though the best practice for structuring the model library remains an area for further research, the chosen structure did prove to be both intuitive for further development and flexible enough to accommodate the use of COTS libraries. Within the powertrain development process an attempt is made to separate the controller modelling from the physical plant modelling. This is in keeping with the object-oriented development philosophy. Control functionality is included in subsystem models but as a separate object that can be more easily modified and replaced, and further allowing the physical plant model to be used separately or exported to external software. Third party models were integrated into the developed HEV library at both a component and a subsystem level. While the use of COTS models proved beneficial in increasing the fidelity of certain subsystems where the time and effort in acquiring the knowledge and experience to develop similar models is not available, the risks of using third party models must be accounted for and managed. Ultimately the developed Dymola powertrain models are shown to be valid through comparison of simulation results with existing Simulink simulations that had been validated against dynamometer tests.

Case Study 2: LifeCar Extension, provided an opportunity to test the proposed development method along with a concurrent "real-world" development activity on the LifeCar vehicle. It provided a chance to reiterate the development performed in the initial case study in much the same way as would be done within a spiral type development methodology. This case study shows the suitability of the hierarchical model library structure for reuse and extension for new model development efforts. Further, the development method is used in developing subsystem and powertrain models for comparing power supply options for the HEV as well as powertrain models for comparing 2WD and 4WD vehicle topologies. Apart from affirming observations made in the first case study, new insights into the modelling requirements were seen. In particular the use of a linked database with parameterization data for the model can simplify the reparameterization task, where the use of central databases for exchanging data between the different software in use in automotive companies lends itself well to the benefits of CAE.

The importance of model management considerations, for achieving the goal of more efficient modelling, and ensuring the successful integration of a new modelling method into an already well established industry, are discussed. At the modelling

environment level, there is a need for enforcing model maintenance features, documentation and encryption through management policy. At the system integration level there is a need for an efficient method of communicating and sharing data and requirements between all stakeholders, automating communication between the various systems being used for increased efficiency and data consistency, and managing the risks associated with implementing new technologies and changes to procedures.

This Thesis shows that the proposed modelling method makes use of the OOM properties of inheritance and encapsulation in order to develop reusable, replaceable and ultimately more maintainable models. Further, the iterative nature of the proposed method is shown to be well suited to development within a prototyping development methodology and therefore should be reasonably easy to integrate into current automotive modelling practices. The research has shown that the overall acceptance of the development method is largely up to the support structure provided for it within the management policies.

## 8.1 Future Work

In order to advance this work further, the author suggests that the proposed model development method be tested in conjunction with a pilot project within an automotive manufacturer. In this way it will be possible to make a direct comparison between this method and traditional modelling methods. Gaining direct feedback from model developers, model users and management will highlight any advantages and disadvantages of changing the model management and development methods. Also in this manner it will be possible to better quantify any benefits in terms of development time and effort. All areas that prove hard to integrate within the current development environment should then be revised, taking into consideration the input from the various parties involved.

Specifically, this project should focus on the use of system modelling to convey data between the stakeholders. The application of SysML or similar modelling languages should be investigated with respect to its suitability for sharing model information between stakeholders and also how well these models contribute to the development of the final object-oriented physical models. In this respect, the CAE aspects of the research should focus on the integration of the system models, data

storage and physical models. Ideally, the aim should be to automate as much of the physical model creation process as possible by drawing on the SysML models for the structure, the model library for the subsystems and components and the shared data storage for parameterization of the models. From the modelling perspective, the benefits of different library structures should be investigated here with a focus on how the structure can best make use of and support the CAE and systems modelling activities.

The final stage of this project should look at the application of this method for real-time and HIL simulation studies. This is necessary in order to establish the suitability of object-oriented models for real-time applications. It is important to understand how the models need to be changed for this purpose and whether these changes can be performed as part of the iterative development process. Again it is important to gain feedback on how this compares with current modelling methods in terms of time and effort from a developer point of view and the implications it may have on the model management process.

# References

[1] G. S. King, R. P. Jones, and D. Simner, "A good practice model for implementation of computer-aided engineering analysis in product development," *Journal Of Engineering Design*, vol. 14, no. 3, pp. 315–331, Sept. 2003.

[2] P. Treffinger, M. Baur, T. Braig, H. Dittus, and J. Ungethüm, "Modelling of Alternative Propulsion Concepts Applying Modular Object-Oriented Simulation Techniques," in *Proceedings of the* $23^{rd}$ *International Electric Vehicle Symposium (EVS-23)*, Anaheim, California, USA, $2^{nd} - 5^{th}$ Dec. 2007.

[3] D. W. Gao, C. Mi, and A. Emadi, "Modeling and Simulation of Electric and Hybrid Vehicles," *Proc. IEEE*, vol. 95, no. 4, pp. 729–745, 2007.

[4] R. Wolfson and J. Gower, "The Role of Computer Modeling and Simulation in Electric and Hybrid Vehicle Research and Development," *IEEE Trans. Veh. Technol.*, vol. 32, no. 1, pp. 62–73, 1983.

[5] M. Jennings and R. Rangan, "Managing complex vehicle system simulation models for automotive system development," *J. Comput. Info. Sci. Eng.*, vol. 4, no. 4, pp. 372–378, 2004.

[6] P. Tsou, L. Rose, J. Che, A. Zaremba, and M. Jennings, "Challenges of Creating Complex Integrated Vehicle System Models," in *Proceedings of MathWorks Automotive Conference (MAC 2007)*. Dearborn, Michigan: MathWorks, $19^{th} - 20^{th}$ June 2007.

[7] C. Schyr and K. Gschweitl, "Model-based Development and Calibration of Hybrid Powertrains," in *SAE Technical Papers*, no. 2007-01-0285. Society of Automotive Engineers, 2007.

[8] S. Khoshafian and R. Abnous, *Object Orientation: Concepts, Analysis and*

*Design, Languages, Databases, Graphical User Interfaces, Standards*, $2^{nd}$ ed. New York: John Wiley & Sons, Inc., 1995.

[9] J. Eborn, L. Pedersen, C. Haugstetter, and S. Ghosh, "System Level Dynamic Modeling of Fuel Cell Power Plants," in *Proceedings of the 2003 American Control Conference*, vol. 3, $4^{th} - 6^{th}$ June 2003, pp. 2024–2029.

[10] J. Tobolář, M. Otter, and T. Bünte, "Modelling of Vehicle Powertrains with the Modelica PowerTrain Library," in *Dynamisches Gesamtsystemverhalten von Fahrzeugantrieben*, 2007, pp. 204–216.

[11] R. Fellini, N. Michelena, P. Papalambros, and M. Sasena, "Optimal Design of Automotive Hybrid Powertrain Systems," in *Proceedings of EcoDesign '99: First International Symposium on Environmentally Conscious Design and Inverse Manufacturing*, $1^{st} - 3^{rd}$ Feb. 1999, pp. 400–405.

[12] R. Sinha, C. J. J. Paredis, V.-C. Liang, and P. K. Khosla, "Modeling and simulation methods for design of engineering systems," *J. Comput. Info. Sci. Eng.*, vol. 1, no. 1, pp. 84–91, 2001.

[13] L. Han, C. J. J. Paredis, and P. K. Khosla, "Object-Oriented Libraries of Physical Components in Simulation and Design," in *Proceedings of 2001 Summer Computer Simulation Conference*, Orlando, Florida, $15^{th} - 19^{th}$ July, 2001.

[14] Toyota Motor Corporation, "Toyota Hybrid System THS II," Last accessed: Oct. 2010. [Online]. Available: http://www.toyota.co.jp/en/tech/environment/ths2/SpecialReports_12.pdf

[15] G. Rizzoni, L. Guzzella, and B. M. Baumann, "Unified Modeling of Hybrid Electric Vehicle Drivetrains," *IEEE/ASME Trans. Mechatronics*, vol. 4, no. 3, pp. 246–257, Sept. 1999.

[16] J. Larsson and P. Krus, "Concepts for Multi-Domain Modelling and Simulation," in *Proceedings of The Seventh Scandinavian International Conference on Fluid Power (SICFP 2001)*, Linköping, Sweden, $30^{th}$ May $-1^{st}$ June, 2001.

[17] P. Le Marrec, C. Valderrama, F. Hessel, A. Jerraya, M. Attia, and O. Cayrol, "Hardware, software and mechanical cosimulation for automotive applications," in *Proceedings of the $9^{th}$ International Workshop on Rapid System Prototyping*, $3^{rd} - 5^{th}$ June 1998, pp. 202–206.

[18] Y. Gao and M. Ehsani, "A Torque and Speed Coupling Hybrid Drivetrain – Architecture, Control, and Simulation," *IEEE Trans. Power Electron.*, vol. 21, no. 3, pp. 741–748, May 2006.

[19] M. Ehsani, Y. Gao, S. E. Gay, and A. Emadi, *Modern Electric, Hybrid Electric, and Fuel Cell Vehicles: Fundamentals, Theory and Design.* Boca Raton, Florida: CRC Press LLC, 2005.

[20] C. C. Chan, "The State of the Art of Electric, Hybrid, and Fuel Cell Vehicles," *Proc. IEEE*, vol. 95, no. 4, pp. 704–718, Apr. 2007.

[21] Y. Gao and M. Ehsani, "Systematic Design of Fuel Cell Powered Hybrid Vehicle Drive Train," in *SAE Technical Papers*, no. 2001-01-2532. Society of Automotive Engineers, 2001.

[22] J.-H. Jung, Y.-K. Lee, J.-H. Joo, and H.-G. Kim, "Power Control Strategy for Fuel Cell Hybrid Electric Vehicles," in *SAE Technical Papers*, no. 2003-01-1136. Society of Automotive Engineers, 2003.

[23] W. Gao, "Performance Comparison of a Fuel Cell-Battery Hybrid Powertrain and a Fuel Cell-Ultracapacitor Hybrid Powertrain," *IEEE Trans. Veh. Technol.*, vol. 54, no. 3, pp. 846–855, May 2005.

[24] K.-W. Suh and A. G. Stefanopoulou, "Effects of Control Strategy and Calibration on Hybridization Level and Fuel Economy in Fuel Cell Hybrid Electric Vehicle," in *SAE Technical Papers*, no. 2006-01-0038. Society of Automotive Engineers, 2006.

[25] A. B. Lovins and D. R. Cramer, "Hypercars®, hydrogen, and the automotive transition," *Int. J. Vehicle Design*, vol. 35, no. 1/2, pp. 50–85, 2004.

[26] J. Marco, N. D. Vaughan, H. Spowers, and M. D. McCulloch, "Modelling the Acceleration and Braking Characteristics of a Fuel-Cell Electric Sports Vehicle Equipped with an Ultracapacitor," *Proc. IMechE Part D: Journal of Automobile Engineering*, vol. 221, no. 1, pp. 67–81, 2007.

[27] J. Marco, N. D. Vaughan, H. Spowers, and M. D. McCulloch, "The Importance of Whole System Design to the Viability of Fuel Cell Vehicles," in *Proceedings of the* $6^{th}$ *International Conference on Materials for Lean Weight Vehicles*, University of Warwick, UK, $7^{th} - 8^{th}$ Dec., 2005.

[28] A. Emadi, K. Rajashekara, S. S. Williamson, and S. M. Lukic, "Topological Overview of Hybrid Electric and Fuel Cell Vehicular Power System Architectures and Configurations," *IEEE Trans. Veh. Technol.*, vol. 54, no. 3, pp. 763–770, May 2005.

[29] J. Erjavec and J. Arias, *Hybrid, Electric & Fuel-Cell Vehicles.* Clifton Park, NY, USA: Thomson Delmar Learning, 2007.

[30] S. S. Williamson, S. M. Lukic, and A. Emadi, "Comprehensive Drive Train Efficiency Analysis of Hybrid Electric and Fuel Cell Vehicles Based on Motor-Controller Efficiency Modeling," *IEEE Trans. Power Electron.*, vol. 21, no. 3, pp. 730–740, May 2006.

[31] A. Boyali, M. Demirci, T. Acarman, L. Güvenç, O. Tür, H. Uçarol, B. Kiray, and E. Özatay, "Modeling and Control of a Four Wheel Drive Parallel Hybrid Electric Vehicle," in *Proc. of the 2006 IEEE International Conference on Control Applications*, Munich, Germany, $4^{th} - 6^{th}$ Oct., 2006, pp. 155–162.

[32] M. Larsen, S. De La Salle, and D. Reuter, "A Reusable Control System Architecture for Hybrid Powertrains," in *SAE Technical Papers*, no. 2002-07-2808. Society of Automotive Engineers, 2002.

[33] I. Husain, *Electric and Hybrid Vehicles: Design Fundamentals.* New York: CRC Press, 2003.

[34] Toyota Motor Corporation, "Hybrid Synergy Drive," Last accessed: Oct. 2010. [Online]. Available: http://www.hybridsynergydrive.com/en/start.html

[35] Honda Motor Company, "Honda Civic Hybrid," Last accessed: Oct. 2010. [Online]. Available: http://world.honda.com/CIVICHYBRID/

[36] T. J. E. Miller, "Optimal Design of Switched Reluctance Motors," *IEEE Trans. Ind. Electron.*, vol. 49, no. 1, pp. 15–27, 2002.

[37] Switched Reluctance Drives Ltd., "Automotive SR Drive Technology," Last accessed: Oct. 2010. [Online]. Available: http://www.srdrives.co.uk/automotive.shtml

[38] J. Marco, "Electrical Architectures for Hybrid Vehicles: Implications for Modelling and Control," in *Proceedings of the 2008 International Conference on Control (UKACC)*, University of Manchester, UK, $2^{nd} - 4^{th}$ Sept., 2008.

[39] R. McGee, F. Syed, S. Hunter, and D. Ramaswamy, "Power Control for the Escape and Mariner Hybrids," in *SAE Technical Papers*, no. 2007-01-0282. Society of Automotive Engineers, 2007.

[40] L. Guzzella and A. Amstutz, "CAE Tools for Quasi-Static Modeling and Optimization of Hybrid Powertrains," *IEEE Trans. Veh. Technol.*, vol. 48, no. 6, pp. 1762–1769, Nov. 1999.

[41] J. Van Mierlo, P. Van den Bossche, and G. Maggetto, "Models of energy sources for EV and HEV: fuel cells, batteries, ultracapacitors, flywheels and engine-generators," in *Journal of Power Sources*, vol. 128, no. 1, 2004, pp. 76–89.

[42] C. Yunpeng, S. Xiaomin, and J. Peifa, "Forward-facing Modeling for an Electric Vehicle Powertrain," in *The 20th International Electric Vehicle Symposium (EVS-20)*, Long Beach, CA, Nov. 2003.

[43] J. Van Mierlo and G. Maggetto, "Vehicle simulation program: a tool to evaluate hybrid power management strategies based on an innovative iteration algorithm," in *Proc. IMechE Part D: Journal of Automobile Engineering*, vol. 215, no. 9, 2001, pp. 1043–1052.

[44] T. Markel, A. Brooker, T. Hendricks, V. Johnson, K. Kelly, B. Kramer, M. O'Keefe, S. Sprik, and K. Wipke, "Advisor: a systems analysis tool for advanced vehicle modeling," *Journal of Power Sources*, vol. 110, no. 2, pp. 255 – 266, Aug. 2002.

[45] B. K. Powell, K. E. Bailey, and S. R. Cikanek, "Dynamic modeling and control of hybrid electric vehicle powertrain systems," *IEEE Control Syst. Mag.*, vol. 18, no. 5, pp. 17–33, 1998.

[46] K. B. Wipke, M. R. Cuddy, and S. D. Burch, "Advisor 2.1: a user-friendly advanced powertrain simulation using a combined backward/forward approach," *IEEE Trans. Veh. Technol.*, vol. 48, no. 6, pp. 1751–1761, Nov. 1999.

[47] E. Cacciatori, "Advanced Control Concepts for a Parallel Hybrid Powertrain with Infinitely Variable Transmission," Ph.D. dissertation, Cranfield University, Cranfield, 2007.

[48] J. Van Mierlo, G. Maggetto, and P. Van den Bossche, "Simulation methodologies for innovative vehicle drive systems," in *11th International Power Electronics and Motion Control Conference (EPE-PEMC'2004)*, Riga, Latvia, $2^{nd} - 4^{th}$ Sept. 2004.

[49] S. Wilkins and M. Lampérth, "The Development of an Object-Oriented Tool for the Modeling and Simulation of Hybrid Powertrains for Vehicular Applications," in *SAE Technical Papers*, no. 2003-01-2249. Society of Automotive Engineers, 2003.

[50] T. Markel and K. Wipke, "Modeling grid-connected hybrid electric vehicles using advisor," in *Proc. of the* $16^{th}$ *Annual Battery Conf. Applications and Advances*, 2001, pp. 23–29.

[51] M. Tiller, P. Bowles, and M. Dempsey, "Development of a Vehicle Modeling Architecture in Modelica," in *Proceedings of the* $3^{rd}$ *International Modelica Conference*, Linköping, Sweden, $3^{rd} - 4^{th}$ Nov. 2003, pp. 75–86.

[52] G. Looye, M. Thümmel, M. Kurze, M. Otter, and J. Bals, "Nonlinear Inverse Models for Control," in *Proceedings of the* $4^{th}$ *International Modelica Conference*, Hamburg-Harburg, Germany, March 7-8 2005, pp. 267–279.

[53] M. Thummel, M. Otter, and J. Bals, "Control of robots with elastic joints based on automatic generation of inverse dynamics models," in *Proc. IEEE/RSJ Int Intelligent Robots and Systems Conf*, vol. 2, 2001, pp. 925–930.

[54] J. Bals, G. Hofer, A. Pfeiffer, and C. Schallert, "Object-Oriented Inverse Modelling of Multi-Domain Aircraft Equipment Systems and Assessment with Modelica," in *Proc. of the* $3^{rd}$ *International Modelica Conference*, Linköping, Sweden, $3^{rd} - 4^{th}$ Nov. 2003, pp. 377–384.

[55] C. E. Newman, J. J. Batteh, and M. Tiller, "Spark-Ignited-Engine Cycle Simulation in Modelica," in *Proceedings of the* $2^{nd}$ *International Modelica Conference*, Oberpfaffenhofen, Germany, $18^{th} - 19^{th}$ Mar., 2002, pp. 133–142.

[56] M. Amrhein and P. T. Krein, "Dynamic Simulation for Analysis of Hybrid Electric Vehicle System and Subsystem Interactions, Including Power Electronics," *IEEE Trans. Veh. Technol.*, vol. 54, no. 3, pp. 825–836, May 2005.

[57] C. Schweiger, M. Dempsey, and M. Otter, "The PowerTrain Library: New Concepts and New Fields of Applications," in *Proceedings of the* $4^{th}$ *International Modelica Conference*, Hamburg-Harburg, Germany, $7^{th} - 8^{th}$ Mar. 2005, pp. 457–466.

[58] P. Fritzson, *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1.* John Wiley & Sons, Inc. (Wiley-IEEE Press), 2004.

[59] J. D. Lambert, *Numerical methods for ordinary differential systems: the initial value problem.* New York, NY, USA: John Wiley & Sons, Inc., 1991.

[60] H. Elmqvist, D. Brück, S. E. Mattsson, H. Olsson, and M. Otter, "Dymola - Dynamic Modeling Laboratory," Dynasim AB, Ideon Science Park, Lund, Sweden, User's Manual Dymola 7.3 vol 1 and 2, 2009.

[61] F. E. Cellier, "Object-oriented modeling: Means for dealing with system complexity," in *Proc. of the* $15^{th}$ *Benelux Meeting on Systems and Control*, Mierlo, The Netherlands, 1996, pp. 53–64.

[62] M. Wetter and C. Haugstetter, "Modelica versus trnsys - a comparison between an equation-based and a procedural modeling language for building energy simulation," in *Proceedings of the* $2^{nd}$ *National Conference of IBPSA-USA (SimBuild 2006)*, Cambridge, MA, USA., 2006.

[63] A. Eriksson and B. Jacobson, "Modular modelling and simulation tool for evaluation of powertrain performance," *International Journal of Vehicle Design*, vol. 21, no. 2/3, pp. 175–189, Oct. 2004.

[64] M. Tiller, P. Bowles, H. Elmqvist, D. Brück, S. E. Mattson, A. Möller, H. Olsson, and M. Otter, "Detailed Vehicle PowertrainModeling in Modelica," in *Modelica Workshop 2000*, Lund, Sweden, $23^{rd} - 24^{th}$ Oct. 2000.

[65] R. M. Schupbach and J. C. Balda, "Comparing DC-DC Converters for Power Management in Hybrid Electric Vehicles," in *Proc. IEMDC'03 Electric Machines and Drives Conference IEEE International*, vol. 3, 2003, pp. 1369–1374.

[66] A. F. Burke, "Batteries and Ultracapacitors for Electric, Hybrid, and Fuel Cell Vehicles," *Proc. IEEE*, vol. 95, no. 4, pp. 806–820, April 2007.

[67] A. Sciarretta and L. Guzzella, "Control of Hybrid Electric Vehicles - A Survey of Optimal Energy-Management Strategies," *IEEE Control Syst. Mag.*, vol. 27, no. 2, pp. 60–70, Apr. 2007.

[68] E. Cacciatori, N. D. Vaughan, and J. Marco, "Energy Management Strategies for a Parallel Hybrid Electric Powertrain: Fuel Economy Optimisation with Driveability Requirements," in *IET Hybrid Vehicle Conference*, University of Warwick, Coventry, UK, $12^{th} - 13^{th}$ Dec., 2006, pp. 157–172.

[69] J. Larminie and J. Lowry, *Electric Vehicle Technology Explained.* John Wiley and Sons, Ltd, 2003.

[70] M. Koot, J. T. B. A. Kessels, B. de Jager, W. P. H. Heemels, P. P. J. van den Bosch, and M. Steinbuch, "Energy Management Strategies for Vehicular Electric Power Systems," *IEEE Trans. Veh. Technol.*, vol. 54, no. 3, pp. 771–782, May 2005.

[71] M. Ceraolo, A. di Donato, and G. Franceschi, "A General Approach to Energy Optimisation of Hybrid-Electric Vehicles," *IEEE Trans. Veh. Technol.*, 2007, accepted for future publication Vehicular Technology.

[72] X. Wei, "Modeling and Control of a Hybrid Electric Drivetrain for Optimum Fuel Economy, Performance and Driveability," Ph.D. dissertation, The Ohio State University, Department of Mechanical Engineering, Columbus, Ohio, 2004.

[73] M. Grotjahn, L. Quernheim, and S. Zemke, "Modelling and identification of car driveline dynamics for anti-jerk controller design," in *2006 IEEE International Conference on Mechatronics*, July, 2006, pp. 131–136.

[74] J. Fredriksson, H. Weiefors, and B. Egardt, "Powertrain Control for Active Damping of Driveline Oscillations," in *Vehicle System Dynamics*, vol. 37, no. 5, 2002, pp. 359–376.

[75] M. Dempsey, S. Biggs, and N. Dixon, "Simulating driveability using Dymola and Modelica," in *Proc. of the* $4^{th}$ *International Modelica Conference*, Hamburg, Germany, $7^{th} - 8^{th}$ Mar. 2005.

[76] H. Tummescheit, "Design and Implementation of Object-Oriented Model Libraries using Modelica," Ph.D. dissertation, Department of Automatic Control, Lund Institute of Technology, Lund, Aug. 2002.

[77] Z. G. Xu, J. H. Frazer, and M. X. Tang, "Novel design methodology supporting product life-cycle design," *Computers in Industry*, vol. 49, no. 3, pp. 253 – 265, 2002.

[78] G. Booch, *Object-oriented analysis and design with applications*, $2^{nd}$ ed. Benjamin/Cummings, 1994, in Cranfield library.

[79] K.-S. Hong, K.-J. Yang, and K.-I. Lee, "Object-Oriented Modeling for Gasoline Engine and Automatic Transmission Systems," *Computer Applications in Engineering Education*, vol. 7, no. 2, pp. 107–119, 1999.

[80] K. J. Åström, H. Elmqvist, and S. E. Mattsson, "Evolution Of Continuous-Time Modeling And Simulation," in *Proc. of the* $12^{th}$ *European Simulation Multiconference (ESM'98).* Manchester, UK: Society for Computer Simulation International, $16^{th} - 19^{th}$ June, 1998, pp. 9–18.

[81] B. Zupančič and A. Sodja, "Object oriented modelling of variable envelope properties in buildings," *WSEAS Trans. Sys. Ctrl.*, vol. 3, no. 12, pp. 1046–1056, Dec. 2008.

[82] M. Tiller, *Introduction to Physical Modeling with Modelica.* Springer, 2001.

[83] H. Elmqvist and S. E. Mattsson, "An Introduction To The Physical Modeling Language Modelica," in *Proceedings of the* $9^{th}$ *European Simulation Symposium (ESS '97)*, Passau, Germany, $19^{th} - 23^{rd}$ Oct. 1997.

[84] Modelica and the Modelica Association, "Modeling of Complex Physical Systems," Last accessed: Oct. 2010. [Online]. Available: http://www.modelica.org/

[85] C. Maffezzoni, L. Ferrarini, and E. Carpanzano, "Object-oriented models for advanced automation engineering," *Control Engineering Practice*, vol. 7, no. 8, pp. 957 – 968, 1999.

[86] T. Pulecchi, F. Casella, and M. Lovera, "Object-oriented modelling for spacecraft dynamics: Tools and applications," *Simulation Modelling Practice and Theory*, vol. 18, no. 1, pp. 63–86, 2010.

[87] M. K. Yu, R. L. Anthony, and D. Castillo, "The application of object-oriented modeling and simulation to aircraft supportability," in $9^{th}$ *AIAA Computing*

*in Aerospace Conference*, no. AIAA-93-4480-CP, San Diego, CA (USA), $19^{th}$ – $21^{st}$ Oct. 1993, pp. 121–128.

[88] P. Sahlin, L. Eriksson, P. Grozman, H. Johnsson, E. Shapovalov, and M. Vuolle, "Will equation-based building simulation make it? experiences from the introduction of ida indoor climate and energy," in *Proceedings of* $8^{th}$ *International IBPSA Conference, International Building Performance Simulation Association*, Eindhoven, The Netherlands, 2003, pp. 1147–1154.

[89] P. Bunus and P. Fritzson, "A Debugging Scheme for Declarative Equation Based Modeling Languages," in *Proc. of the* $4^{th}$ *International Symposium on Practical Aspects of Declarative Languages (PADL '02).* London, UK: Springer-Verlag, 2002, pp. 280–298.

[90] A. Rousseau, P. Sharer, and F. Besnier, "Feasability of Reusable Vehicle Modeling: Application to Hybrid Vehicles," in *SAE Technical Papers*, no. 2004-01-1618. Society of Automotive Engineers, 2004.

[91] F. E. Cellier, H. Elmqvist, and M. Otter, *Modeling from Physical Principles*, W. S. Levine, Ed. CRC Press, 1995.

[92] C. B. Moler, *Numerical Computing with Matlab.* Society for Industrial Mathematics, January 2004. [Online]. Available: http://www.mathworks.com/moler/chapters.html

[93] R. LeVeque, *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems.* Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2007.

[94] J. J. Ramos, M. À. Piera, and I. Serra, "The use of physical knowledge to guide formula manipulation in system modelling," *Simulation Practice and Theory*, vol. 6, no. 3, pp. 243–254, 1998.

[95] M. R. Blaha and J. R. Rumbaugh, *Object-Oriented Modeling and Design with UML*$^{TM}$, $2^{nd}$ ed. Upper Saddle River, NJ.: Pearson Prentice Hall, 2005.

[96] P. A. Fishwick, "Toward a Convergence of Systems and Software Engineering," Department of Computer and Information Science and Engineering, University of Florida, Florida, USA, Tech. Rep. REP-1996-203, Jan. 1996. [Online]. Available: http://www.cise.ufl.edu/tr/REP-1996-203/

[97] I. Graham, *Object Oriented Methods*, $2^{nd}$ ed. Wokingham, UK: Addison-Wesley Publishing Co., Inc., 1994.

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

[99] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, $2^{nd}$ ed., ser. (The Addison-Wesley Object Technology Series). Addison-Wesley Professional, May 2005.

[100] R. G. Fichman and C. F. Kemerer, "Object-oriented and conventional analysis and design methodologies," *Computer*, vol. 25, no. 10, pp. 22–39, 1992.

[101] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, $5^{th}$ ed., ser. McGraw-Hill Series in Computer Science. London: McGraw-Hill Higher Education, 2001.

[102] M. Priestley, *Practical Object-oriented Design with UML, 2nd Ed.*, 2nd ed. McGraw-Hill Publishing Co., Mar. 2000.

[103] Object Management Group, Inc., "Documents associated with UML Version 2.3," Last accessed: Oct. 2010. [Online]. Available: http://www.omg.org/spec/UML/2.3/

[104] S. Rachuri, Y.-H. Han, S. C. Feng, F. Wang, R. D. Sriram, K. W. Lyons, and U. Roy, "Object-oriented representation of electro-mechanical assemblies using uml," in *Proc. IEEE Int. Assembly and Task Planning Symp*, 2003, pp. 228–234.

[105] J. Marco and N. D. Vaughan, "Integration of Architectural Modelling using the SysML within the Traditional Automotive CACSD Process," in *UKACC International Conference on Control*, Coventry University, UK, Sept. 2010.

[106] T. A. Johnson, C. J. J. Paredis, R. Burkhart, and J. M. Jobe, "Modeling Continuous System Dynamics in SysML," in *2007 ASME International Mechanical Engineering Congress and Exposition (IMECE 2007)*, Seattle, Washington, USA, $11^{th} - 15^{th}$ Nov., 2007.

[107] Object Management Group, Inc., "Documents associated with SysML Version 1.2," Last accessed: Oct. 2010. [Online]. Available: http://www.omg.org/spec/SysML/1.2/

[108] Y. Grobshtein, V. Perelman, E. Safra, and D. Dori, "Systems Modeling Languages: OPM Versus SysML," in *Proc. Int. Conf. Systems Engineering and Modeling ICSEM '07*, Haifa, Israel, 2007, pp. 102–109.

[109] W. Schamai, P. Fritzson, C. Paredis, and A. Pop, "Towards Unified System Modeling and Simulation with ModelicaML: Modeling of Executable Behavior Using Graphical Notations," in *Proceedings of the* $7^{th}$ *Modelica Conference*, Como, Italy, $20^{th} - 22^{nd}$ Sept. 2009, pp. 612–621.

[110] W. Royce, "Managing the development of large software systems," in *Proc. IEEE Wescon*, August 1970, pp. 1–9.

[111] D. M. Buede, *The engineering design of systems: models and methods.* John Wiley & Sons, Inc., 2000.

[112] C. Wong, "A successful software development," *IEEE Trans. Softw. Eng.*, no. 6, pp. 714–727, 1984.

[113] B. Boehm, "A spiral model of software development and enhancement," *SIGSOFT Softw. Eng. Notes*, vol. 11, no. 4, pp. 14–24, 1986.

[114] M. Alavi, "An assessment of the prototyping approach to information systems development," *Commun. ACM*, vol. 27, no. 6, pp. 556–563, 1984.

[115] K. Forsberg and H. Mooz, "The relationship of system engineering to the project cycle," in *Proceedings of the National Council for System Engineering (NCOSE) Conference*, Chattanooga, TN, Oct. 1991, pp. 57–65.

[116] J. M. Colombi and R. G. Cobb, "Application of systems engineering to rapid prototyping for close air support," *Defense A R Journal*, vol. 16, no. 3, pp. 284–303, Oct. 2009.

[117] R. Harrison, A. A. West, and L. J. Lee, "Lifecycle Engineering of Future Automation Systems in the Automotive Powertrain Sector," in *Proceedings of the IEEE International Conference on Industrial Informatics*, 2006, pp. 305–310.

[118] G. H. Fisher, "Model-based systems engineering of automotive systems," in *Proc. of the* $17^{th}$ *Digital Avionics Systems Conf. (DASC)*, vol. 1. AIAA/IEEE/SAE, 1998.

[119] M. Broy, "Automotive Software and Systems Engineering," in *Proc. of the* $3^{rd}$ *ACM and IEEE Int. Conf. Formal Methods and Models for Co-Design (MEMOCODE '05)*, 2005, pp. 143–149.

[120] S. A. Sheard, "The Frameworks Quagmire, A Brief Look," in *Proceedings of the* $7^{th}$ *Annual Symposium of the International Council on Systems Engineering (INCOSE)*, Los Angeles, CA, 1997.

[121] S. A. Sheard and J. G. Lake, "Systems Engineering Standards and Models Compared," in *Proceedings of the* $8^{th}$ *Annual Symposium of the International Council on Systems Engineering (INCOSE)*, Vancouver, Canada, 1998, pp. 589–605.

[122] *ANSI/EIA Processes for Engineering a System*, EIA Std. 632, 1999.

[123] *ISO/IEC/IEEE Systems and Software Engineering - System Life Cycle Processes*, IEEE Std. 15 288-2008, 2008.

[124] J. N. Martin, "Overview of the EIA 632 standard: processes for engineering a system," in *Proc. of the* $17^{th}$ *AIAA/IEEE/SAE Digital Avionics Systems Conf.*, vol. 1, 1998.

[125] *ISO/IEC/IEEE Systems and Software Engineering - Software Life Cycle Processes*, IEEE Std. 12 207-2008, 2008.

[126] *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, IEEE Std. 1471-2000, 2000.

[127] *IEEE Guide–Adoption of ISO/IEC TR 24748-1:2010 Systems and Software Engineering–Life Cycle Management–Part 1: Guide for Life Cycle Management*, IEEE Std. 24 748-1-2011, 2011.

[128] T. Williams, *Modelling Complex Projects*. London, UK: John Wiley and Sons, Ltd, 2002.

[129] Technical Specialist: Functional Modelling, "Interviews with CAE Simulation Group at Jaguar Cars Ltd." Personal communications, 2010.

[130] J. Marco and E. Cacciatori, "The Use of Model Based Design Techniques in the Design of Hybrid Electric Vehicles," in *The* $3^{rd}$ *IET International Automotive Electronics Conference*, University of Warwick, June 2007.

[131] J. McManus and T. Wood-Harper, "Understanding the Sources of Information Systems Project Failure," *Journal of the Institute of Management Services*, vol. 51, no. 3, pp. 38–43, 2007.

[132] G. Ferretti and P. Rocco, "Special issue on modular physical modelling," *Mathematical and Computer Modelling of Dynamical Systems*, vol. 12, no. 1, pp. 1–3, Feb. 2006.

[133] M. Barth, M. Strube, A. Fay, P. Weber, and J. Greifeneder, "Object-oriented engineering data exchange as a base for automatic generation of simulation models," in *Proc. of the* $35^{th}$ *Annual Conference of IEEE Industrial Electronics (IECON '09)*, 2009, pp. 2465–2470.

[134] W. Gao, S. Neema, J. Gray, J. Picone, S. Porandla, S. Musunuri, and J. Mathews, "Hybrid Powertrain Design Using a Domain-Specific Modeling Environment," in *Proc. IEEE Conf. Vehicle Power and Propulsion*, Sept. 2005, pp. 423–429.

[135] J. R. Josephson, B. Chandrasekaran, M. Carroll, N. Iyer, B. Wasacz, G. Rizzoni, Q. Li, and D. A. Erb, "An Architecture for Exploring Large Design Spaces," in *Proceedings of the National Conference of the American Association for Artificial Intelligence*, Madison, WI, 1998, pp. 143–150.

[136] J. Batteh and M. Tiller, "Implementation of an extended vehicle model architecture in modelica for hybrid vehicle modeling: Development and applications," in *Proceedings of the* $7^{th}$ *International Modelica Conference*, Como, Italy, $20^{th} - 22^{nd}$ Sept. 2009, pp. 823–832.

[137] J. Marco and N. D. Vaughan, "The control-oriented design and simulation of a high voltage bus management strategy for use within hybrid electric vehicles," in *International Journal of Vehicle Systems Modelling and Testing*, vol. 2, no. 4, 2007, pp. 345–368.

[138] P. Sawyer, I. Sommerville, and S. Viller, "Requirements process improvement through the phased introduction of good practice," in *Software Process - Improvement and Practice*, 1997, pp. 31–44.

[139] D. Leffingwell, "Calculating the Return on Investment from More Effective Requirements Management," in *American Programmer*, vol. 10, no. 4, Apr. 1997, pp. 13–16.

[140] M. Dempsey, M. Gäfvert, P. Harman, C. Kral, M. Otter, and P. Treffinger, "Coordinated automotive libraries for vehicle system modelling," in *Proc. of the* $5^{th}$ *International Modelica Conference*, Vienna, Austria, $4^{th} - 5^{th}$ Sept. 2006, pp. 33–41.

[141] Dassault Systémes and DLR, "PowerTrain Library," Last accessed: Oct. 2010. [Online]. Available: http://www.3ds.com/products/catia/portfolio/dymola/model-libraries/

[142] J. V. Gragger, "SmartElectricDrives Library," arsenal research, Giefinggasse 2, A 1210 Vienna, Austria, Users Manual Version 1.3.1, 2009. [Online]. Available: http://www.arsenal.ac.at/modelica/

[143] MODELISAR Project, "Functional Mock-up Interface," Last accessed: Oct. 2010. [Online]. Available: http://www.functional-mockup-interface.org/

[144] S. Robinson, R. E. Nance, R. J. Paul, M. Pidd, and S. J. Taylor, "Simulation model reuse: definitions, benefits and obstacles," *Simulation Modelling Practice and Theory*, vol. 12, no. 7-8, pp. 479–494, 2004, simulation in Operational Research.

[145] T. Monks, S. Robinson, and K. Kotiadis, "Model reuse versus model development: Effects on credibility and learning," in *Proceedings of the 2009 Winter Simulation Conference*, 2009, pp. 767–778.

[146] T. J. Woolmer and M. D. McCulloch, "Analysis of the yokeless and segmented armature machine," in *IEEE International Electric Machines & Drives Conference, 2007 (IEMDC '07)*, vol. 1, May 2007, pp. 704 –708.

[147] B. Boehm and C. Abts, "Cots integration: plug and pray?" *Computer*, vol. 32, no. 1, pp. 135–138, 1999.

[148] D. Garlan, R. Allen, and J. Ockerbloom, "Architectural mismatch: why reuse is so hard," *IEEE Softw.*, vol. 12, no. 6, pp. 17–26, 1995.

[149] SourceForge.net, "Open Source Software," Last accessed: Oct. 2010. [Online]. Available: http://sourceforge.net/about

[150] Modelica and the Modelica Association, "Modelica Standard Library," Last accessed: Oct. 2010. [Online]. Available: http://www.modelica.org/libraries/Modelica

[151] C. Pelchen, C. Schweiger, and M. Otter, "Modeling and Simulating the Efficiency of Gearboxes and of Planetary Gearboxes," in *Proc. of the* $2^{nd}$ *International Modelica Conference (Modelica'2002)*, Oberpfaffenhofen, Germany, $18^{th} - 19^{th}$ Mar., 2002, pp. 257–266.

[152] J. Larminie and A. Dicks, *Fuel Cell Systems Explained*, $2^{nd}$ ed. John Wiley and Sons, Ltd, 2003.

[153] Dassault Systémes and AIT Austrian Institute of Technology, "SmartElectricDrives Library," Last accessed: Oct. 2010. [Online]. Available: http://www.3ds.com/products/catia/portfolio/dymola/model-libraries/

[154] J. Marco and N. D. Vaughan, "A Classical Control Approach to the Power Management of an All-Electric Hybrid Vehicle," *International Journal of Vehicle Systems Modelling and Testing*, vol. 4, no. 1-2, pp. 55–78, 2009.

[155] W. Gao, N. Zhang, and H. P. Du, "A half-car model for dynamic analysis of vehicles with random parameters," in *Proceedings of the 5th Australasian Congress on Applied Mechanics (ACAM 2007)*, Brisbane, Australia, $10^{th} - 12^{th}$ Dec. 2007, pp. 595–600.

[156] G. Genta, *Motor Vehicle Dynamics: Modeling and Simulation*, 2nd ed. Singapore: World Scientific, 2003.

[157] G. Rill, "First order tire dynamics," in *Proceedings of the* $3^{rd}$ *European Conference on Computational Mechanics Solids, Structures and Coupled Problems in Engineering*. Lisbon, Portugal: Springer Netherlands, $5^{th} - 8^{th}$ June 2006, pp. 1–9. [Online]. Available: http://dx.doi.org/10.1007/1-4020-5370-3_776

[158] G. Rill, "Wheel Dynamics," in *Proceedings of the XII International Symposium on Dynamic Problems of Mechanics (DINAME 2007)*, Ilhabela, SP, Brazil, $26^{th}$Feb.-$2^{nd}$ Mar. 2007.

[159] E. Bakker, H. B. Pacejka, and L. Lidner, "A New Tire Model with an Application in Vehicle Dynamics Studies," in *SAE Technical Papers*, no. 890087. Society of Automotive Engineers, 1989.

[160] H. B. Pacejka and E. Bakker, "The magic formula tyre model," *Vehicle System Dynamics: International Journal of Vehicle Mechanics and Mobility*, vol. 21, no. 1 Supplement 1, pp. 1–18, Nov. 1992.

[161] M. Short, M. Pont, and Q. Huang, "Simulation of Vehicle Longitudinal Dynamics," Embedded Systems Laboratory, University of Leicester, Technical report ESL 04/01, 2004.

[162] J.-M. Zhang, B.-Y. Song, and G. Sun, "An Advanced Control Method for ABS Fuzzy Control System," in *2008 International Conference on Intelligent Computation Technology and Automation (ICICTA)*, vol. 1, $20^{th} - 22^{nd}$ Oct. 2008, pp. 845–849.

[163] B. Olsen, S. W. Shaw, and G. Stépán, "Nonlinear Dynamics of Vehicle Traction," in *Vehicle System Dynamics*, vol. 40, no. 6, 2003, pp. 377–399.

[164] H. Yeo, S. Hwang, and H. Kim, "Regenerative braking algorithm for a hybrid electric vehicle with CVT ratio control," in *Proc. IMechE Part D: Journal of Automobile Engineering*, vol. 220, no. 11, 2006, pp. 1589–1600.

[165] Woodbank Communications Ltd., "Electropaedia," Last accessed: Oct. 2010. [Online]. Available: http://www.mpoweruk.com/index.htm

[166] P. Schmidt and T. Rehm, "Notch Filter Tuning for Resonant Frequency Reduction in Dual Inertia Systems," in *Proceedings of the* $34^{th}$ *IEEE IAS Annual Meeting*, vol. 3, Phoenix, $3^{rd} - 7^{th}$ Oct. 1999, pp. 1730–1734.

[167] M. Pidd, "Five Simple Principles of Modelling," in *Proceedings of the 1996 Winter Simulation Conference*, 1996, pp. 721–728.

[168] L. Poeti, J. Marco, and N. D. Vaughan, "HEV Energy Storage Evaluation Using an Object-Oriented Modelling Methodology," in *Proceedings of the* $10^{th}$ *International Symposium on Advanced Vehicle Control (AVEC 10)*, Loughborough, UK, $22^{nd} - 26^{th}$Aug. 2010.

[169] J. R. Miller, "Development of Equivalent Circuit Models for Batteries and Electrochemical Capacitors," in *Proc. of the* $14^{th}$ *Annual Battery Conference on Applications and Advances*, 1999, pp. 107–109.

[170] P. Thounthong, S. Raël, and B. Davat, "Control Strategy of Fuel Cell and Supercapacitors Association for a Distributed Generation System," *IEEE Trans. Ind. Electron.*, vol. 54, no. 6, pp. 3225–3233, 2007.

[171] G. Standley, J. Marco, and B. Cho, "The impact of vehicle usage and recharging infrastructure on the energy storage requirements of a plug-in hybrid

electric vehicle." *Int. J. Electric and Hybrid Vehicles*, vol. 2, no. 3, pp. 222–239, 2010.

[172] LiFeBATT Ltd., "ENERGY CELLS," Last accessed: Oct. 2010. [Online]. Available: http://www.lifebatt.co.uk/energy_cells.html

[173] M. A. Roscher and D. U. Sauer, "Dynamic electric behavior and open-circuit-voltage modeling of lifepo4-based lithium ion secondary batteries," *Journal of Power Sources*, vol. 196, no. 1, pp. 331–336, 2011.

[174] J. Auer, G. Sartorelli, and J. Miller, "A Gatekeeper Energy Management Strategy for ECVT Hybrid Vehicle Propulsion Utilising Ultracapacitors," in *Proc. IET Hybrid Vehicle Conference 2006*, Warwick University, Coventry, UK, Dec. 2006, pp. 79–90.

[175] N. Jinrui, S. Fengchun, and R. Qinglian, "A Study of Energy Management System of Electric Vehicles," in *IEEE Vehicle Power and Propulsion Conference (VPPC '06)*, $6^{th} - 8^{th}$ Sept., 2006, pp. 1–6.

[176] Z. Han, Z. Yuan, T. Guangyu, C. Quanshi, and C. Yaobin, "Optimal Energy Management Strategy for Hybrid Electric Vehicles," in *SAE Technical Papers*, no. 2004-01-0576. Society of Automotive Engineers, 2004.

[177] V. A. Shah, S. G. Karndhar, R. Maheshwari, P. Kundu, and H. Desai, "An energy management system for a battery ultracapacitor hybrid electric vehicle," in *Proc. of the* $4^{th}$ *International Conference on Industrial and Information Systems (ICIIS)*, 2009, pp. 408–413.

[178] L. C. Rosario and P. C. K. Luk, "Implementation of a modular power and energy management structure for battery - ultracapacitor powered electric vehicles," in *Proceedings of IET Hybrid Vehicle Conference*, 2006, pp. 141–156.

[179] G. Paganelli, G. Ercole, A. Brahma, Y. Guezennec, and G. Rizzoni, "General Supervisory Control Policy for the Energy Optimization of Charge-Sustaining Hybrid Electric Vehicles," *JSAE Review*, vol. 22, pp. 511–518, 2001.

[180] M. E. Fayad, W.-T. Tsai, and M. L. Fulghum, "Transition to object-oriented software development," *Commun. ACM*, vol. 39, no. 2, pp. 108–121, 1996.

[181] A. P. J. Breunese, J. F. Broenink, J. L. Top, and J. M. Akkermans, "Libraries of reusable models: Theory and application," *SIMULATION*, vol. 71, no. 1, pp. 7–22, July 1998.

[182] B. Lohmann and W. Marquardt, "On the systematization of the process of model development," *Computers & Chemical Engineering*, vol. 20, no. Supplement 1, pp. S213–S218, 1996, european Symposium on Computer Aided Process Engineering-6.

[183] J. Fulem, "ICCT evaluation of vehicle simulation tools," Ricardo, Tech. Rep., Aug. 2009.

[184] A. T. Bahill and S. J. Henderson, "Requirements Development, Verification, and Validation Exhibited In Famous Failures," *Systems Engineering*, vol. 8, no. 1, pp. 1–13, 2005.

[185] M. C. Hause and F. Thom, "An integrated mda approach with sysml and uml," in *Proc. 13th IEEE Int. Conf. Engineering of Complex Computer Systems ICECCS 2008*, 2008, pp. 249–254.

[186] M. Weber and J. Weisbrod, "Requirements Engineering in Automotive Development: Experiences and Challenges," *IEEE Softw.*, vol. 20, no. 1, pp. 16–24, Jan./Feb. 2003.

[187] ITEA2 Program, "Information Technology for European Advancement," Last accessed: Oct. 2010. [Online]. Available: http://www.itea2.org/public/project_leaflets/MODELISAR_profile_oct-08.pdf

[188] A. P. Sage and W. B. Rouse, Eds., *Handbook of Systems Engineering and Management*, $2^{nd}$ ed. Hoboken, New Jersey: John Wiley & Sons, Inc., 2009.

[189] Y. Tao, "A study of software development project risk management," in *Proc. Int. Seminar Future Information Technology and Management Engineering FITME '08*, 2008, pp. 309–312.

[190] A. Kossiakoff and W. N. Sweet, *Systems Engineering Principles and Practice*. John Wiley & Sons, Inc., 2003.

[191] Y. H. Kwak and J. Stoddard, "Project risk management: lessons learned from software development environment," *Technovation*, vol. 24, no. 11, pp. 915–920, 2004.

[192] M. Khalifa and J. M. Verner, "Drivers for Software Development Method Usage," *IEEE Trans. Eng. Manag.*, vol. 47, no. 3, pp. 360–369, 2000.

[193] C. K. Riemenschneider, B. C. Hardgrave, and F. D. Davis, "Explaining Software Developer Acceptance of Methodologies: A Comparison of Five Theoretical Models," *IEEE Trans. Softw. Eng.*, vol. 28, no. 12, pp. 1135–1145, 2002.

[194] B. C. Hardgrave and R. A. Johnson, "Toward an information systems development acceptance model: the case of object-oriented systems development," *IEEE Trans. Eng. Manag.*, vol. 50, no. 3, pp. 322–336, 2003.

[195] J. Schmuller, *Sams Teach Yourself UML in 24 Hours*, 3rd ed. Sams Publishing, 2004.

# Appendix A

# Glossary

## A.1 Object-Oriented Terminology

This section serves as a reference with short descriptions of the most common concepts used in object-oriented programming [58, 76, 195]. Examples of how these concepts are applied within the modelling environment are provided in Sections 3.3, 5.2 and 6.2.

1. **Abstraction** – Abstraction is a key activity in developing an object-oriented program. It is the skill of simplifying the detail of a given problem into the essential features which can then be used to create class descriptions. This process helps to break down the problem (programming task) into more manageable (and reusable) portions. For example a car can be be abstracted into a container of objects such as engine, transmission and wheels. When defining the car it is not necessary to know how the internal components are defined, only how they should be used together. Depending on the purpose of the program, different levels of abstraction may be employed. For instance, a transportation class could be a higher abstraction containing any means of transportation such as bike, car or train and the transmission could be abstracted further giving lower level objects such as a gear, clutch and torque converter.

2. **Object** – Objects can be seen as the fundamental component of an object-oriented program because it is their interaction that defines what the program

does at execution. In addition objects have features which describe the object and what it can do. These features are commonly referred to as attributes and operations. Following from the previous example, a tyre might be an object with *radius* and *weight* as attributes and *slip angle* as an operation.

3. **Class** – A class is sometimes described as the blueprint from which an object is created. It is a grouping of similar types of objects and a direct result of the abstraction process. The class should represent a real world abstraction from the problem domain e.g. bank account in financial program or engine in car model. The attributes and operations common to similar objects are defined in the class. A visual way of representing classes during the design of an object-oriented system is by means of the Unified Modeling Language (UML) class diagrams as shown in Figure A.1.

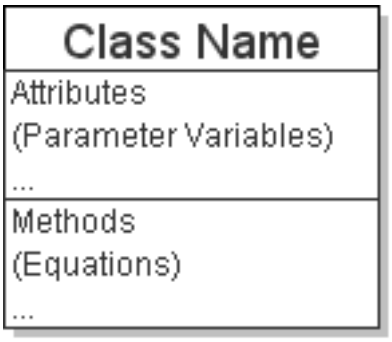



Figure A.1: Example UML representation of a class

4. **Abstract Class** – This refers to a class whose level of abstraction is too high for an object to be created directly from it. In other words it is an incomplete class often used as a parent class (see inheritance).

5. **Instance** – When an object is created through the use of a class, this process is called instantiation. In other words an object is an instance of a class. In a car class we can create an instance of an engine class or an engine object with relevant parameters for that car.

6. **Encapsulation** – Encapsulation describes the ability to hide the internal workings (code) of a class from external objects. For example, in order for the driver to see the speed of the car, it is not necessary to know what sensors are being used to calculate it and how this is being done. This is an important factor in the maintainability of a program or system, since the external

objects will not be affected by changes to encapsulated code e.g. the method of determining the speed can be changed without affecting the driver.

7. **Modularity** – The process of abstraction and the use of classes provides a framework for modularity to the developed system since it enforces a logical partitioning of the problem into more manageable and self-sufficient modules, thereby also benefiting the maintainability of the system.

8. **Inheritance** – It is possible to form a child class by inheriting from a parent class. A child class (or subclass) will automatically acquire all the code (attributes and operations) that the parent class had. The parent class is a more abstract class and its code can be reused by all its child classes. In our example, the car class was a child of the vehicle class. It should be noted that the child class can also have additional attributes and operations that the parent class did not have.

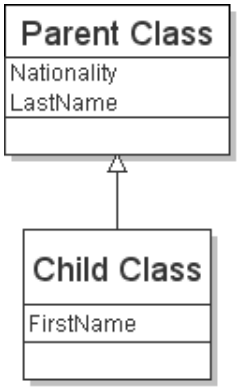



Figure A.2: Example UML representation of a child class inheriting from a parent class

9. **Multiple Inheritance** – This describes the ability of a class to inherit from two or more parent classes. In general, these parent classes should themselves be unrelated. For example, an electrical machine class can inherit both a class that defines a torque connection and a class that defines an electrical connection.

10. **Aggregation** – Is the term used to describe the grouping of the abstracted objects to form a more complex class. In other words, a class can be made up of instances of other classes. Therefore a car class can contain an instance of a chassis class, an instance of a driveline class and so on.

11. **Declarative programming** – defines "what" should be accomplished without specifying the "how".

12. **Imperative programming** – describes "how" to accomplish a task.

## A.2 Common Stages of Development Life-Cycle Models

Descriptions for the common stages are as follows [8, 95]:

1. **Specification of requirements**. During this stage questions need to be posed about the systems functionality such as, "what must it do?", "who will use the system?" and "how will it be used?" Essentially this stage calls for the developers and end-users to communicate their needs in order to refine a formal set of requirements.

2. **System analysis**. Here the requirements from the previous step are analysed in a hierarchical manner so that the system can be described in terms of classes, objects and their relationships. Noting that the focus at this point is on *what* the systems goals are and **not** *how* they are achieved.

3. **Design**. At this point the developer takes the analysis models and decides on *how* the system will achieve its set of goals. In general this is done in two phases with the first phase decomposing the model into smaller subsystems and the second phase focusing on designing specific classes. This stage builds on the analysis stage in order to produce a complete specification of all subsystems and components, their functioning and relationships.

4. **Implementation**. This is the last stage in producing a working system. It is only at this stage that considerations are made for any particular programming language. Now that a complete set of functional and design requirements have been defined, these can be implemented using the features provided by the chosen language.

5. **Testing**. Though the system produced by this stage is functional and should be largely error free if the prior stages were diligently carried out, it must be tested in its entirety in order to verify that it meets the original requirements.

It should be noted that testing of subsystems, components and specific code is performed throughout the development process.

6. **Maintenance**. Complex system designs need to evolve in order to meet future needs of the users, it is important to maintain the system even after development has stopped and it has been passed on to the end-user. Maintenance can take the form of debugging errors not found during the testing stage, restructuring the system, adding enhancements and customization.

# Bibliography

An, F., Stodolsky, F., and Santini, D. (1999). Hybrid Options for Light-Duty Vehicles. In *SAE Technical Papers*, number 1999-01-2929. Society of Automotive Engineers.

Barth, M., Strube, M., Fay, A., Weber, P., and Greifeneder, J. (2009). Object-oriented engineering data exchange as a base for automatic generation of simulation models. In *Proc. of the* $35^{th}$ *Annual Conference of IEEE Industrial Electronics (IECON '09)*, pages 2465–2470.

Baxter, J., Carter, P., Erekson, T., and Todd, R. (2004). UltraCapacitor Power for a Drag Racecar. In *SAE Technical Papers*, number 2004-01-3500. Society of Automotive Engineers.

Brahma, A., Guezennec, Y., and Rizzoni, G. (2000). Optimal Energy Management in Series Hybrid Electric Vehicles. In *American Control Conference*, pages 60–64, Chicago, Illinois.

Broy, M., Pretschner, A., Salzmann, C., and Stauner, T. (2006). Software-Intensive Systems in the Automotive Domain: Challenges for Research and Education. In *SAE Technical Papers*, number 2006-01-1458. Society of Automotive Engineers.

Carson II, J. S. (2004). Introduction to modeling and simulation. In *Proceedings of the 2004 Winter Simulation Conference*, pages 9–16.

Carson II, J. S. (2005). Introduction to modeling and simulation. In *Proceedings of the 2005 Winter Simulation Conference*, pages 16–23.

Chang, L. (1993). Recent Developments of Electric Vehicles and Their Propulsion Systems. *IEEE Aerosp. Electron. Syst. Mag.*, 8(12):3–6.

Chu, L., Wang, Q., Liu, M., and Li, J. (2004). Study of the Control Algorithm of Series Power Train for Fuel Cell Transit Bus. In *SAE Technical Papers*, number 2004-01-2607. Society of Automotive Engineers.

Ciesla, C., Keribar, R., and Morel, T. (2000). Engine/Powertrain/Vehicle Modeling Tool Applicable to All Stages of the Design Process. In *SAE Technical Papers*, number 2000-01-0934. Society of Automotive Engineers.

Cloutier, R. J. and Verma, D. (2007). Applying the Concept of Patterns to Systems Architecture. *Systems Engineering*, 10(2):138–154.

Cole, R. (2006). The Changing Role of Requirememtns and Architecture in Systems Engineering. In *Proc. 2006 IEEE/SMC International Conference on System of Systems Engineering*, pages 6–10, Los Angeles, CA, USA.

Dassault Systémes and DLR and Modelon and ATI and Claytex Services Ltd and Ricardo UK Ltd (2008). VehicleInterfaces Library. Last accessed: Oct.

Deur, J., Petric, J., Asgari, J., and Hrovat, D. (2006). Recent Advances in Control-Oriented Modeling of Automotive Power Train Dynamics. *IEEE/ASME Trans. Mechatronics*, 11(5):513–523.

Dolk, D. R. and Konsynski, B. R. (1984). Knowledge Representation for Model Management Systems. *IEEE Trans. Softw. Eng.*, 10(6):619–628.

Dominka, S., Broecker, E., and Schiller, F. (2008). Automated execution of simulation studies demonstrated via a simulation of a car. In *Proceedings of the 2008 Winter Simulation Conference*, pages 2916–2924, Austin, TX (USA).

Dorey, R. E. and Holmes, C. B. (1999). Vehicle Driveability - Its Characterisation and Measurement. In *SAE Technical Papers*, number 1999-01-0949. Society of Automotive Engineers.

Drogies, S. and Bauer, M. (2002). Modeling Road Vehicle Dynamics with Modelica. In *SAE Technical Papers*, number 2002-01-1219. Society of Automotive Engineers.

Dufour, C., Bélanger, J., Ishikawa, T., and Uemura, K. (2005). Advances in Real-Time Simulation of Fuel Cell Hybrid Electric Vehicles. In *Proc. of the* $21^{st}$ *Electric Vehicle Symposium (EVS-21)*, Monte Carlo, Monaco.

Erkkinen, T. and Snyder, M. F. (1999). Reducing Software Maintenance Costs By Designing in Reliability. In *Proceedings of the Advanced Simulation Technology and Training Conference (SIMTECT '99)*, Melbourne, Vic. Simulation Industry Association of Australia (SIAA).

Fairley, R. E. (1974). Continuous system simulation languages: Design principles and implementation techniques. In *Proceedings of the ACM SIGPLAN symposium on Very high level languages*, pages 73–81, New York, NY, USA. ACM.

Fishwick, P. A. (1998). A taxonomy for simulation modeling based on programming language principles. *IIE Transactions*, 30(9):811–820.

Friedman, D. J., Lipman, T., Eggbert, A., Ramaswamy, S., and Hauer, K.-H. (2000). Hybridization: Cost and Efficiency Comparisons for PEM Fuel Cell Vehicles. In *SAE Technical Papers*, number 2000-01-3078. Society of Automotive Engineers.

Fröberg, A. and Nielsen, L. (2003). Efficient drive cycle simulation. *IEEE Trans. Veh. Technol.*, (99):1–11. Accepted for future publication Vehicular Technology.

Gao, L., Dougal, R., and Liu, S. (2003). Active power sharing in hybrid battery/capacitor power sources. In *Applied Power Electronics Conference and Exposition, 2003. APEC '03. Eighteenth Annual IEEE*, volume 1, pages 497–503vol.1.

Gao, W., Neema, S., Gray, J., Picone, J., Porandla, S., Musunuri, S., and Mathews, J. (2005). Hybrid Powertrain Design Using a Domain-Specific Modeling Environment. In *Proc. IEEE Conf. Vehicle Power and Propulsion*, pages 423–429.

Gao, Y. (2010). Research on the Rule of Evolution of Software Development Process Model. In *Proceedings of the* $2^{nd}$ *IEEE Int. Information Management and Engineering (ICIME) Conf.*, pages 466–470.

Garlan, D., Allen, R., and Ockerbloom, J. (2009). Architectural mismatch: Why reuse is still so hard. *IEEE Softw.*, 26(4):66–69.

Gebhard, B. and Rappl, M. (2000). Requirements Management for Automotive Systems Development. In *SAE Technical Papers*, number 2000-01-0716. Society of Automotive Engineers.

Grimm, K. (2003). Software Technology in an Automotive Company - Major Challenges. In *Proc.* $25^{th}$ *International Conference on Software Engineering (ICSE'03)*, pages 498–503.

Guimarães, L. R. and Vilela, P. R. S. (2005). Comparing software development models using CDM. In *Proceedings of the 6th conference on Information technology education*, pages 339–347, Newark, NJ, USA. ACM.

Harel, D. and Rolph, S. (1989). Modeling and Analyzing Complex Reactive Systems: The Statemate Approach. In *Proc. AIAA Computers in Aerospace VII Conf.*, Monterey, CA (USA).

Heinecke, H., Schnelle, K.-P., Fennel, H., Bortolazzi, J., Lundh, L., Leflour, J., Maté, J.-L., Nishikawa, K., and Scharnhorst, T. (2004). AUTomotive Open System ARchitecture – An Industry-Wide Initiative to Manage the Complexity of Emerging Automotive E/E Architectures. In *SAE Technical Papers*, number 2004-21-0042. Convergence Transportation Electronics Association, USA.

Hellgren, J. (2002). Modelling of Hybrid Electric Vehicles in Modelica for Virtual Prototyping. In *Proc. of the* $2^{nd}$ *International Modelica Conference (Modelica'2002)*, pages 247–256, Oberpfaffenhofen, Germany.

Hoffmann, H.-P., Sibbald, C., and Sibbald, C. (2009). Systems engineeringthe foundation for success in complex systems development. IBM White paper, Systems engineering best practices. Last accessed: Oct.

IBM (2009). Requirements engineering for the automotive industry. IBM White paper, Product development. Last accessed: Oct.

IEEE (2000). IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. Technical Report IEEE Std 1471-2000, IEEE-SA Standards Board, Piscataway, NJ.

ISE Corporation (2010). ISE Advanced Energy Products. Last accessed: Oct.

Jacobson, B., Fredriksson, J., Hellgren, J., Karlsson, J., Scarpati, J., Templin, P., and Vallejo, H. (2000). Modelica Usage in Automotive Problems at Chalmers. In *Modelica Workshop 2000*, pages 147–152, Lund, Sweden.

Jadhav, A. S. and Sonar, R. M. (2009). Evaluating and selecting software packages: A review. *Inf. Softw. Technol.*, 51(3):555–563.

Kalan, B. A., Lovatt, H. C., Brothers, M., and Buriak, V. (2002). System Design and Development of Hybrid Electric Vehicles. In *Proc. IEEE* $33^{rd}$ *Annual Power Electronics Specialists Conference (PESC'02)*, volume 2, pages 768–772.

Katrašnik, T. (2007). Hybridization of powertrain and downsizing of ic engine - a way to reduce fuel consumption and pollutant emissions -part 1. *Energy Conversion and Management*, 48:1411–1423.

Kendall, I. R. and Jones, R. P. (1999). An investigation into the use of hardware-in-the-loop simulation testing for automotive electronic control systems. *Control Engineering Practice*, 7(11):1343–1356.

Khayyam, H., Kouzani, A. Z., and Hu, E. J. (2008). An Intelligent Energy Management Model for a Parallel Hybrid Vehicle Under Combined Loads. In *IEEE International Conference on Vehicular Electronics and Safety, 2008. (ICVES 2008)*, pages 145–150.

Kozaki, T., Mori, H., Fathy, H. K., and Gopalswamy, S. (2004). Balancing the Speed and Fidelity of Automotive Powertrain Models Through Surrogation. In *Proceedings of (IMECE 2004) International Mechanical Engineering Congress and R&D Expo*, Anaheim, CA, USA.

Krüger, I. H., Nelson, E. C., and Prasad, K. V. (2004). Service-Based Software Development for Automotive Applications. In *SAE Technical Papers*, number 2004-21-0040. Convergence Transportation Electronics Association, USA.

Lai, J.-S. and Nelson, D. J. (2007). Energy Management Power Converters in Hybrid Electric and Fuel Cell Vehicles. *Proc. IEEE*, 95(4):766–777.

Laine, L. and Andreasson, J. (2003). Modelling of Generic Hybrid Electric Vehicles. In *Proceedings of the* $3^{rd}$ *International Modelica Conference*, pages 87–94, Linköping, Sweden.

Lauber, J. and Tischer, C. (2001). Using Patterns to Integrate Views in Open Automotive Systems. In *SAE Technical Papers*, number 2001-01-3396. Society of Automotive Engineers and Messe Düsseldorf.

Law, A. M. (2005). How to build valid and credible simulation models. In *Proceedings of the 2005 Winter Simulation Conference*, pages 24–32.

Lewi, J., Steegmans, E., and De Man, J. (1991). Object-oriented approach to software development, a walk through a number of topics. In *Proc. Advanced Computer Technology 5th Annual European Computer Conf Reliable Systems and Applications CompEuro '91*, pages 626–633.

Lovatt, H. C. and Dunlop, J. B. (2002). Power Transfer in Hybrid Electric Vehicles with Multiple Energy Storage Units. In *PEMD*, pages 171–176, Bath, UK.

Marco, J. and Vaughan, N. D. (2006). Improving the Commercial Viability of Fuel Cell Vehicles Through the Application of Integrated Whole Vehicle Control. In *FISITA 2006* $31^{st}$ *World Automotive Congress*, Yokohama, Japan.

Marco, J. and Vaughan, N. D. (2007). Defining Performance Metrics for Hybrid Electric Vehicles. In *SAE Technical Papers*, number 2007-01-0287. Society of Automotive Engineers.

Marei, M. I., Lambert, S., Pick, R., and Salama, M. M. A. (2005). DC/DC Converters for Fuel Cell Powered Hybrid Electric Vehicle. In *Proc. IEEE Vehicle Power and Propulsion Conference (VPPC'05)*, pages 126–129.

Monti, S., Nesci, W., Angellotti, S., Schellino, C., Seminara, M., and Wuesthenagen, R. (2008). Configuration and change management of the outcomes of an automotive engine control model based software design process. In *Proc. 32nd Annual IEEE Int. Computer Software and Applications COMPSAC '08*, pages 1065–1069.

Moreton, P. (2000). *Industrial Brushless Servomotors*. Newnes, Oxford.

Musselman, K. (1994). Guidelines for simulation project success. In *Proceedings of the 1994 Winter Simulation Conference*, pages 88–95.

Naylor, S. M., Pickert, V., and Atkinson, D. J. (2006a). Fuel Cell Drive Train Systems - Driving Cycle Evaluation of Potential Topologies. In *Proc. IEEE Vehicle Power and Propulsion Conference (VPPC'06)*.

Naylor, S. M., Pickert, V., and Atkinson, D. J. (2006b). Fuel Cell Drive Train Topologies - Computer Analysis of Potential Systems. In *The* $3^{rd}$ *IET International Conference on Power Electronics, Machines and Drives (PEMD'06)*, pages 398–403, Dublin, Ireland.

Naylor, S. M., Pickert, V., and Atkinson, D. J. (2006c). Optimization of Compressor Power Supply and Control Systems for Automotive Fuel Cell Drive Train Applications. In *Proc. IEEE Vehicle Power and Propulsion Conference (VPPC '06)*.

Neill, C. J. and Laplante, P. A. (2003). Requirements engineering: the state of the practice. *IEEE Softw.*, 20(6):40–45.

Otter, M., Dempsey, M., and Schlegel, C. (2000). Package PowerTrain: A Modelica library for modeling and simulation of vehicle power trains. In *Proceedings of the* $1^{st}$ *Modelica Workshop*, pages 23–32, Lund, Sweden.

Ouyang, M., Xu, L., Li, J., Lu, L., Gao, D., and Xie, Q. (2006). Performance comparison of two fuel cell hybrid buses with different powertrain and energy management strategies. *Journal of Power Sources*, 163:467–479.

Peng, D., Zhang, Y., liang Yin, C., and wu Zhang, J. (2007). Design of Hybrid Electric Vehicle Braking Control System with Target Wheel Slip Ratio Control. In *SAE Technical Papers*, number 2007-01-1515. Society of Automotive Engineers.

Pisu, P. and Rizzoni, G. (2007). A Comparative Study of Supervisory Control Strategies for Hybrid Electric Vehicles. *IEEE Trans. Control Syst. Technol.*, 15(3):506–518.

Poeti, L., Marco, J., and Vaughan, N. D. (2009). Object Oriented Plant Models for HEV Controller Development. In *SAE Technical Papers*, number 2009-01-0148. Society of Automotive Engineers.

Rangan, R. M. and Chadha, B. (2001). Engineering information management to support enterprise business processes. *J. Comput. Info. Sci. Eng.*, 1(1):32–40.

Reichart, G. and Haneberg, M. (2004). Key Drivers for a Future System Architecture in Vehicles. In *SAE Technical Papers*, number 2004-21-0025. Convergence Transportation Electronics Association, USA.

Saeks, R., Cox, C. J., Mays, P. R., and Murray, J. J. (2002). Adaptive Control of a Hybrid Electric Vehicle. *IEEE Trans. Intell. Transp. Syst.*, 3(4):213–234.

Sahlin, P. (2000). The methods of 2020 for building envelope and hvac systems simulation– will the present tools survive? In *Proceedings of CIBSE National Conference (CIBSE2000)*, Dublin, Ireland. Chartered Institution of Building Services Engineers.

Sahlin, P., Bring, A., and Kolsaker, K. (1995). Future Trends of the Neutral Model Format (NMF). In *Proc.* $4^{th}$ *IBPSA World Congress "Building Simulation '95"*, pages 537–544, Madison, WI. Int. Building Performance Simulation Association.

Sahlin, P., Bring, A., and Sowell, E. F. (1996). The Neutral Model Format for Building Simulation, Version 3.02. Technical report, Department of Building Sciences, The Royal Institute of Technology, Stockholm, Sweden.

Sanz, R. and Zalewski, J. (2003). Pattern-Based Control Systems Engineering. *IEEE Control Syst. Mag.*, 23(3):43–60.

Sedigh-Ali, S., Ghafoor, A., and Paul, R. (2001). Software Engineering Metrics for COTS-Based Systems. *Computer*, 34(5):44–55.

Simic, D., Giuliani, H., Kral, C., and Gragger, J. V. (2006). Simulation of Hybrid Electric Vehicles. In *The 5th International Modelica Conference (Modelica'2006)*, pages 25–31, Vienna, Austria.

Snyder, M. F. (1999). The Importance and Value of Real-Time Simulation. In *Proceedings of the Advanced Simulation Technology and Training Conference (SIMTECT '99)*, Melbourne, Vic. Simulation Industry Association of Australia (SIAA).

Stevens, M. B., Mendes, C., Fowler, M., and Fraser, R. A. (2006). Fuel Cell Hybrid Control Strategy Development. In *SAE Technical Papers*, number 2006-01-0214. Society of Automotive Engineers.

Sullo, G. C. (1994). *Object Engineering: Designing Large-Scale, Object-Oriented Systems*. John Wiley & Sons, Inc.

Tiller, M., Tobler, W. E., and Kuang, M. (2002). Evaluating Engine Contributions to HEV Driveline Vibrations. In *Proceedings of the* $2^{nd}$ *International Modelica Conference*, pages 19–24, Oberpfaffenhofen, Germany.

Timmermans, J.-M., Zadora, P., Cheng, Y., Van Mierlo, J., and Lataire, P. (2005). Modelling and Design of Super Capacitors as Peak Power Unit for Hybrid Electric Vehicles. In *Proc. IEEE Conference Vehicle Power and Propulsion*, pages 701–708.

Topp, K. and Weber, J. (2001). Information Technology - A Challenge for Automotive Electronics. In *SAE Technical Papers*, number 2001-01-0029. Society of Automotive Engineers.

Urquia, A. and Dormido, S. (2003). Object-oriented Design of Reusable Model Libraries of Hybrid Dynamic Systems Part One: A Design Methodology. *Math-*

*ematical and Computer Modelling of Dynamical Systems: Methods, Tools and Applications in Engineering and Related Sciences*, 9(1):65–90.

van Genuchten, M. (1991). Why is software late? an empirical study of reasons for delay in software development. *IEEE Trans. Softw. Eng.*, 17(6):582–590.

Wakefield, D. S., Elliott, D. R., Gauvin, J. P. R., and Masys, A. J. (2006). Managing Modelling and Simulation in Canada's Department of National Defence. In *SimTecT 2006 (Simulation Technology and Training Conference)*, Melbourne, Australia. SIAA.

Walker, A., McGordon, A., Hannis, G., Picarelli, A., Breddy, J., Carter, S., Vinsome, A., Jennings, P., Dempsey, M., and Willows, M. (2006). A Novel Structure for Comprehensive HEV Powertrain Modelling. In *Vehicle Power and Propulsion Conference, 2006. VPPC '06. IEEE*, pages 1–5.

Williamson, S. S., Wirasingha, S. G., and Emadi, A. (2006). Comparative Investigation of Series and Parallel Hybrid Electric Drive Trains for Heavy-Duty Transit Bus Applications. In *Proc. Vehicle Power and Propulsion Conference (VPPC '06). IEEE*.

Winkler, D. and Gühmann, C. (2006). Hardware-in-the-Loop simulation of a hybrid electric vehicle using Modelica®/Dymola®. In *The 22nd International Battery, Hybrid and Fuel Cell Electric Vehicle Symposium & Exposition (EVS-22)*, pages 1054–1063, Yokohama, Japan. Japan Automobile Research Institute.

Yang, J., Bauman, J., and Beydoun, A. (2006). Architecture Design and Implementation Issues for Model-Based Automotive Embedded Software Development. In *SAE Technical Papers*, number 2006-01-3519. Society of Automotive Engineers.

Yang, W. Z., Xie, S. Q., Ai, Q. S., and Zhou, Z. D. (2008). Recent development on product modelling: a review. *International Journal of Production Research*, 46(21):6055–6085.

Zhu, Y., Chen, Y., Tian, G., Wu, H., and Chen, Q. (2004). A four-step method to design an energy management strategy for hybrid vehicles. In *Proceedings of the 2004 American Control Conference*, volume 1, pages 156–161.

Zupančič, B., Karba, R., Atanasijevič-Kunc, M., and Musič, J. (2008). Continuous systems modelling education – causal or acausal approach? In *Proceedings of the*

*ITI 2008* $30^{th}$ *Int. Conf. on Information Technology Interfaces*, pages 803–808, Cavtat, Croatia.