Chapter 11
Industrial Application of Co-modelling and Co-simulation Technology

Marcel Verhoef and Peter Gorm Larsen

11.1 Introduction

This chapter provides an overview of three industrial applications that have been carried out using the Crescendo co-modelling and co-simulation technology. The models have been developed by industry users and this presentation also includes a summary of the achievements and main lessons learned from the work performed in an industrial context. The applications include

- a dredging excavator system,
- a document handling system and
- a self-balancing scooter, the "ChessWay", as introduced in Chap. 7.

The applications reported in this chapter have been produced by three different companies as case studies for the DESTECS project [28] under which the Crescendo technology was originally developed. First, we provide a short overview of the companies involved:

Verhaert New Products & Services N.V. Verhaert is an integrated product development center delivering innovation, development and engineering services for state-of-the-art high-technology products. Verhaert is based in Kruibeke, Belgium, and is active in many technology development programmes leading to innovative hardware, software and devices. For more information, we refer to http://www.verhaert.com.

M. Verhoef (✉)
Chess WISE, Haarlem, The Netherlands
e-mail: Marcel.Verhoef@chess.nl

P.G. Larsen
Aarhus University, Aarhus, Denmark
e-mail: pgl@eng.au.dk

Table 11.1 Case study context

Application	Self-balancing scooter	Document inserting system	Dredging excavator
Company	CHESS	NEOPOST	VERHAERT
Disciplines	Mechanical, control, electrical, software	Mechanical, control, electrical, software	Mechanical, control, electrical, software
Fault source	Design faults	Error handling	Operator error
Challenge	Manage complexity	Concurrent design	Product robustness
Improvement	Reliability analysis	MIL simulations	Design exploration
Purpose	Trustworthy design	Raise product quality	Reduce product TTM
Approach	DE-first, CT-first and contract-first	DE-first	CT-first
Prior knowledge	VDM, 20-sim	20-sim	–

MIL model in the loop, *TTM* time to market

Neopost Technologies B.V. Neopost is one of the two world leaders in small- to middle-range mail handling equipment manufacturing. Its portfolio includes machines for flexible packing of multiformat documents, automated handling of incoming high volume mail and supporting information systems. Neopost Technologies B.V. is based in Drachten, the Netherlands, with a 75-head R&D force, which forms the "Document Systems" Competence Centre of the Neopost Group. For more information, see http://www.neopost-technologies.com.

Chess WISE B.V. and **Chess iX B.V.** The Chess group forms a design and development centre with core competences in both hardware and software. They also take operational responsibility and perform life-cycle maintenance of the electronic products and systems that they develop. Chess is based in Haarlem, the Netherlands, with a multidisciplinary engineering team of approximately 50 engineers. See http://www.chess.eu and http://www.chess-ix.com.

The applications described in this chapter are very different in nature and they use the alternative approaches introduced in Chap. 8. The case studies have been carefully selected to provide a range of embedded systems applications with different forms of complexity, involving engineering heterogeneity (so that collaborative approaches are of interest) and all having the need to provide a predictable level of fault tolerance. They were chosen to represent a state-of-the-art innovative design problem, but they also intended to be recognisable and acceptable to the industry at large, in order to ensure impact. They were proposed and taken on by individual partners in order to reduce risks and optimise resourcing but also to expose different working practices. Thus, they complement each other and illustrate the variety of situations in which the Crescendo technology is applicable. An overview is provided in Table 11.1.

Unfortunately, we are not able to present the full co-models from the case studies for commercial reasons. However, we are able to provide an overview of the approaches taken, to a certain extent the structure of the co-models and the main findings from the case studies. This chapter first introduces the dredging excavator case study from Verhaert in Sect. 11.2. The Neopost document handling system is presented in Sect. 11.3. For confidentiality reasons, the details of the latter model are replaced by a paper path co-simulation model from the BODERC project [96] that has similar characteristics. Afterwards, Sect. 11.4 introduces the ChessWay application. Finally, Sect. 11.5 provides a summary of the chapter.

11.2 A Dredging Excavator

11.2.1 Case Description and Main Challenges

The Verhaert case study concerns the design of an excavator-based dredging system. The dredging process can be defined as the repositioning of soil from the seafloor for infrastructural or ecological purposes. This process typically involves expensive machinery and complex operator control as well as several days of operation, resulting in expensive and difficult processes. A dredging system is a complex application that involves a thorough logistical planning of the dredging process itself, besides the associated control problem of the elements involved in it.

As a general goal, it is Verhaert's intention to elevate the performance of dredging systems by introducing semi-automated control to improve productivity and also to reduce the down time due to repairs. At a lower (loop control) level, this includes a dynamic response function of the digging resistance to avoid overload of the machine. At a higher (sequence control) level, automation should provide predefined trajectory control (e.g. digging along a straight path) and optimisation of the digging pattern. The ultimate goal of this automation is to design an excavator which can operate completely autonomously.

Currently, the expertise of the operator has a large impact on the performance of the machine. In particular, the resistance encountered by the bucket during its path is crucial. For example, a novice operator tends to apply too much force with the arm, while an experienced operator is able to operate the excavator smoothly. By creating a system that assists the user, optimal operation can always be achieved. The goal of this case study is to investigate an automated control system as described above in order to increase the performance and the uptime of the excavator.

Verhaert based their work on a realistic excavator scale model with three axes of freedom (the *shoulder*, connecting the undercarriage with the boom, the *elbow*, connecting the boom with the stick and finally the *wrist*, connecting the stick and the bucket), which allowed for easy model validation as the setup can be used for realistic tests and measurements under controlled conditions. The scale model, which is shown in Fig. 11.1, is powered by electrical linear actuators instead of hydraulics, which are common in practice. The continuous time model of the excavator case study is introduced in Sect. 11.2.2 and the discrete event model is presented in Sect. 11.2.3. Afterwards Sect. 11.2.4 presents the co-simulation

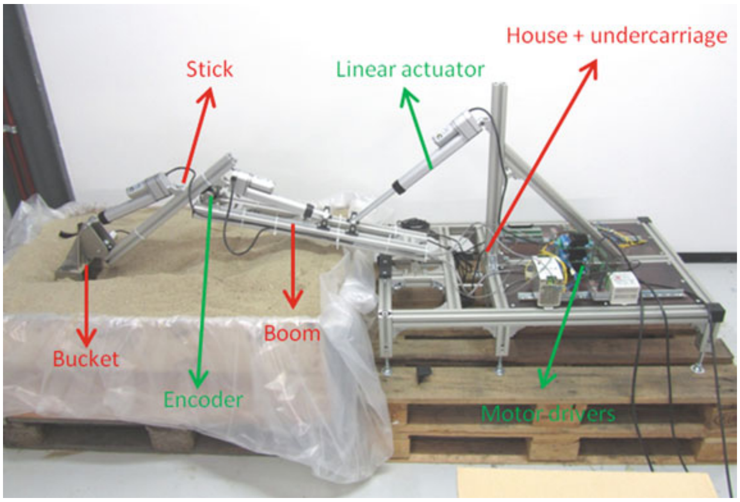

Fig. 11.1 The Verhaert excavator scale model and test setup

analysis conducted on this co-model and finally Sect. 11.2.5 provides an overview of the key results of this study.

11.2.2 The Continuous Time Model

A top-level overview of the CT model of the excavator is presented in Fig. 11.2. This figure uses the block diagram notation but represents a hierarchy of differential equations, as illustrated by the model snippet in Fig. 11.3. The electrical linear actuators are managed using encoders that send motor axle rotation measurements over to the controller. Based on the requested operator inputs, the controller determines the Pulse Width Modulation (PWM)signals to be sent to the power amplifiers driving the motors inside the linear actuators that make the excavator parts (boom, stick and bucket) move.

A special ground model was introduced to investigate the impact of ground material properties on the automated control, by comparing simulations with different ground material properties (e.g. mass density, drag). The ground model describes the forces experienced by the bucket when digging through the ground. Two different forces have been modelled: a constant cut force (velocity independent) that is required to break open the ground and a drag force (velocity-dependent) due to the bucket moving through the ground. The ground consists of several layers (along the z-direction) of different composition. The parameters of these layers (height, cut and drag force, mass density) can be set by reading in a table from a text file. It is also possible to activate an obstacle at a certain position in the x-direction.

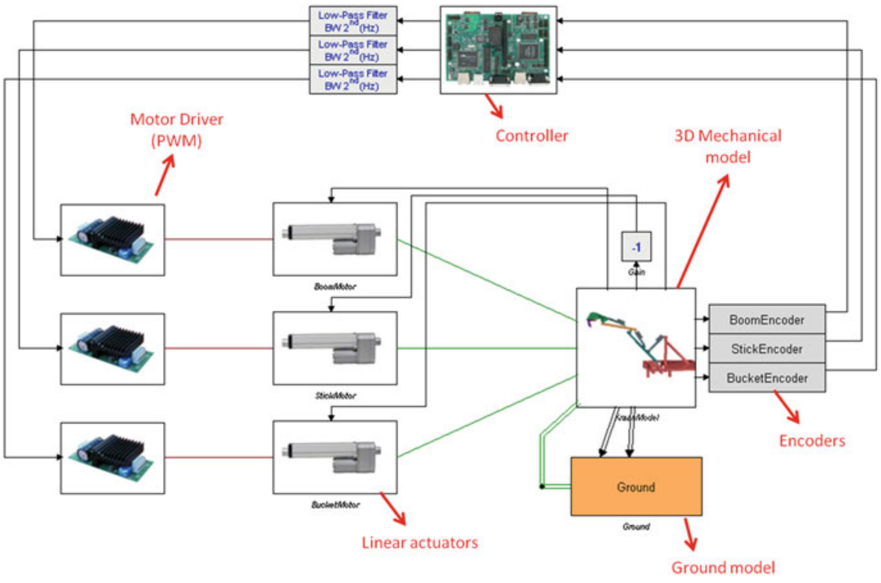

Fig. 11.2 The top-level continuous time model of the excavator

```
equations
// Motor equations
Tm = K * iMotor;
p1.u = K * omega + R * iMotor + 0.0000000001;

p1.i = iMotor + minCurrent* sign(p1.u);

// spindle speed relation
p2.v = omega * i;

// spindle force/torque relation
Tm = Tmax * Fspindel^2 / ( Fspindel^2 + Fx^2 );

Fspindel = p2.F + col;

load  = Tm / i;
overload = (not collision and abs(load) > maxLoad);
```

Fig. 11.3 Part of the 20-sim submodel for a linear actuator

This obstacle has a different composition than the rest of the layer. This feature can be used to model discontinuities, for example, a hard rock in the otherwise soft soil, at which point the excavator will experience a possible overload when trying to dig through it. The ground model also calculates the volume and mass of the ground that is dug up. This mass is then fed back into the mechanical model, to represent the varying mass and inertia of the bucket load while digging. In case of the dredging excavator, buoyancy effects can be added if the bucket is still under water.

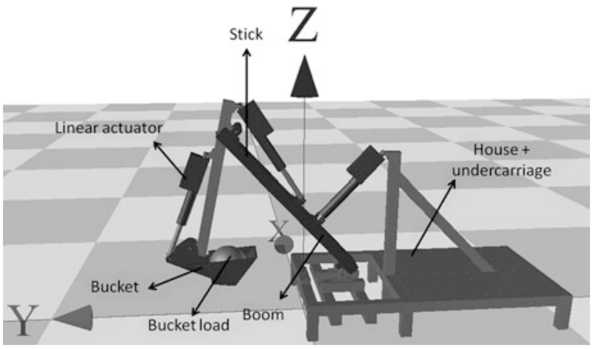

Fig. 11.4 The 3D mechanical model of the excavator

Visualisation techniques can be used in order to get a better understanding of the physical behaviour of the excavator. 20-sim provides a 3D mechanical editor that creates a 3D model with all its associated static properties (such as centre of gravity, inertia and so on). This 3D model is automatically and directly coupled to the differential equations describing the dynamical behaviour of the system. Animation is basically a time-lapse recording of the evolution of those model variables, and this is visualised directly in the 3D model. A screen dump of the excavator animation is shown in Fig. 11.4.

11.2.3 The Discrete Event Model

An overview of the structure of the VDM model of the excavator is shown in Fig. 11.5. The model consists primarily of three VDM classes, each allocated to their own CPU, to denote their independence. All three have a separate control loop running periodically, albeit at different intervals. The *operator* mimics the behaviour of the human operator in terms of using the controls at his disposal: the *joystick* and some *buttons*. The *controller* is the core of the model implementing the feedback control strategy, by reading from *sensors* and writing to *actuators*. The *safety unit* is concerned with independent assessment of the system safety state. It also observes the *sensors* and in case it detects an anomaly, it will shutdown the *controller*.

The main VDM classes *operator*, *controller* and *safety unit* are explained in more detail in the following sections. Also, the key operating modes of the excavator are described, as they are essential to understand the dependability analysis conducted in Sect. 11.2.4.

11.2.3.1 Operator

The `Operator` class is the first of the three principal classes. Although in a physical sense the operator is not part of the controller hardware, the creation of this class allows us to model the interaction between the human operator and the

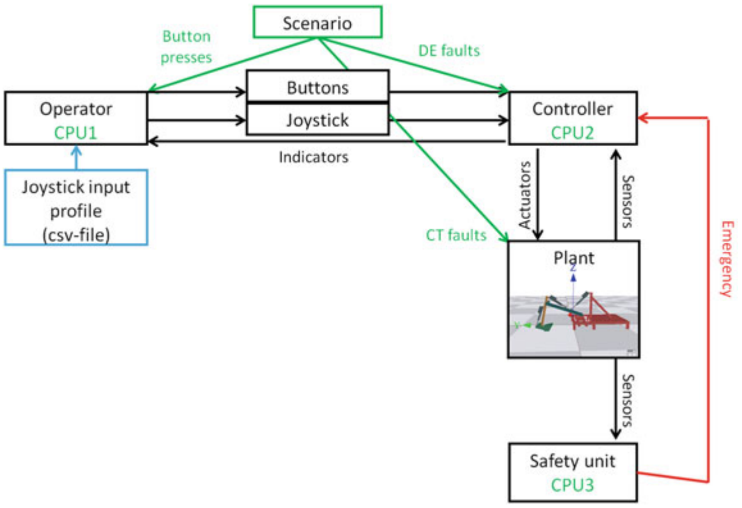

Fig. 11.5 A context diagram of the VDM model of the excavator

system. The operator receives input from the controller by means of indicators (lights and gauges on the dashboard), as well as visual feedback (the positions of the components of the excavator). Based on this feedback, he can manually operate the excavator using buttons and joysticks. The joystick handling at a given time is imported by reading a table from a comma separated text file using the `CSV` standard library.

11.2.3.2 Controller

The second principal class of the VDM model is the `Controller` class. The controller has a total of ten inputs: four buttons (power on/off, emergency on/off, start/stop and manual/assisted); three joysticks (one for each of the three Degrees of Freedom (DoF): boom, stick and bucket); and three sensor values (the angles between each of these components, measured with a relative encoder). Based on a fixed time step (typically 100 Hz), the controller reads these input values, processes them and provides three output signals to the actuators (again, one for each DoF). Using the co-simulation engine, the controller exchanges the sensor and actuator signals with the CT model of the excavator. The relationships from the `Controller` class are shown in a class diagram in Fig. 11.6.

11.2.3.3 Safety Unit

The third and final principal class is the `SafetyUnit`. This class has a thread that runs in parallel with the controller and also reads and processes the sensor values (the sensor signals are split in the software and then sent to both the controller

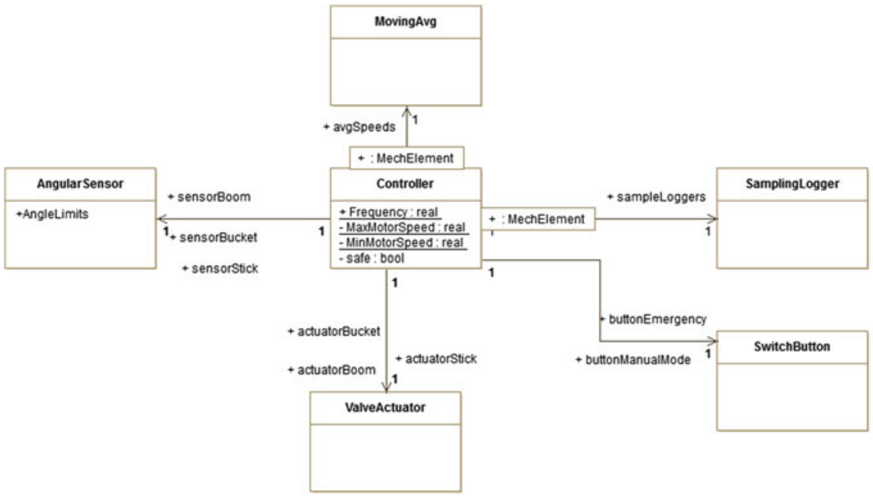

Fig. 11.6 An overview of the relationships from the `Controller` class

and the safety unit). The safety unit is a redundant system that under normal circumstances does not need to take any actions. It acts as a safety mechanism that only comes into action when the controller fails. There can be different reasons for such failures (e.g. a software bug, an unforeseen situation or a hardware failure), but this is irrelevant from the point of view of the safety unit.

One specific function of the safety unit is to prevent the actuators from crashing into their end stops, potentially resulting in breakdown of the entire excavator. Even though the controller contains the intelligence to slow down the motion to a soft stop before running into the end stop, the safety unit always makes an additional check to ensure that a crash does not occur. Since the safety unit only makes simple checks, it requires less processing power and thus runs on a lower-performance processor, but at a higher speed.

When action is required, the safety unit intervenes in several ways simultaneously. Firstly, it triggers the emergency state of the controller, resulting in a power shutdown of the excavator, thereby stopping all motion. This emergency state will only be reached when the controller is still running. An example of this is a controller failure due to an erroneous calculation of the limits. Secondly, the safety unit overrules the output of the controller and thereby stops all motion. Lastly, the safety unit opens three relays, by which power to all actuators is cut off. The system can be restarted by following an explicit reset procedure where the controller has only limited functionality at its disposal until the safe operating zone is reached again.

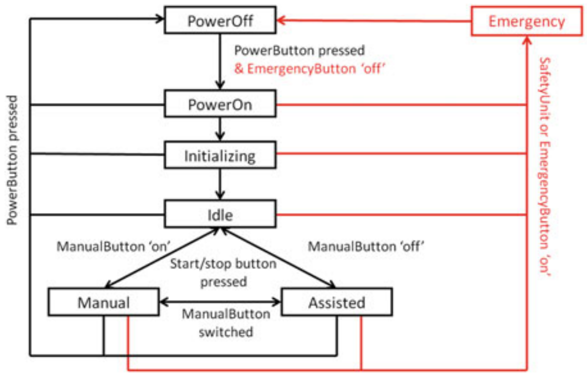

Fig. 11.7 The different states of the controller class

11.2.3.4 Controller States

The controller class has seven different states, described by the informal state diagram in Fig. 11.7. The initial state is "PowerOff", in which the excavator is unpowered. By pressing the "PowerButton", the operator turns on power and the controller goes to the "PowerOn" state and then to the "Initialising" state. This state can be used to perform a homing procedure in which the actuators are moved to their starting position. This position is then used as a reference such that the read out of the relative encoders can be used to measure absolute angles between the components. When the initialisation is done, the controller turns to the "Idle" state. In the "Idle" state, the excavator is powered and the controller is running. By pressing the "Start/Stop button", the controller moves either to the "Manual" or the "Assisted" state, depending on whether the "ManualButton" is switched on or off. When the controller is in either of these two states, switching the "ManualButton" turns the controller from the "Manual" to the "Assisted" state or vice versa. By pressing the "Start/Stop button" again, the controller moves back to the "Idle" state. When the operator pushes the "PowerButton" again, power to the excavator is shut down and the controller returns to the "PowerOff" state.

Besides these normal states, the controller also has an "Emergency" state. This state can either be triggered by the operator by turning on the "EmergencyButton", or by the "SafetyUnit" when a dangerous situation is detected. When the "Emergency" state is triggered, power is shut down and the controller moves to the "PowerOff" state. As long as the "EmergencyButton" is on, power cannot be restored and so the controller remains in the "PowerOff" state.

11.2.3.5 Modes of Operation

In normal operation, the controller runs in either "Manual" or "Assisted" mode. These are the only two modes in which output signals can be sent to the actuators to move the excavator. To prevent overload of the actuators by sudden changes in input

```
speedPercentageAtAngle: MechElement * real * real ==> real
speedPercentageAtAngle(element, angle, speed) ==
(...
 let softLimit = sensors(element).getSoftLimit(useMinAngleLimits),
     hardLimit = sensors(element).getHardLimit(useMinAngleLimits)
 in
 if slowdownLimiter = 1.0
 then ratio := 1.0 - (angle - softLimit) / (hardLimit - softLimit);
 return if hardLimit > softLimit
        then if angle > hardLimit
             then 0.0
             elseif angle > softLimit
             then ratio
             else 1.0
        elseif hardLimit < softLimit
        then if angle < hardLimit
             then 0.0
             elseif angle < softLimit
             then ratio
             else 1.0)
```

Fig. 11.8 A VDM model snippet of the motion limiter

by the operator, the controller smoothes out the input signals from the joysticks. This is done by defining a slope, which limits the maximum rate by which the signal can vary over time. Since the actuators have a limited range, we must also prevent them crashing into their end stops. Therefore, the controller reads in the encoder signals which are then used to determine the angles between the components. By setting a limit to these angles, we can limit the movement of the actuators and prevent these crashes. For both endpoints (beginning and end) three limits are defined. Two of these, the soft and hard limit, are used by the controller. As soon as an actuator passes the soft limit, the controller limits the output signal to this actuator. When the actuator passes the hard limit, the signal is turned to zero. In between both limits, the actuator signal is limited by multiplying the signal with a factor that scales linear between 0 (hard limit) and 1 (soft limit). A VDM model snippet of this function is shown in Fig. 11.8. This limiting is only activated when the actuator is moving towards the end stop. When it is moving away from it, its motion is not limited. The third limit that is set for this range limiting lies behind the hard limit. It is used by the safety unit and when the actuator passes this limit the "Emergency" state is triggered and power to the excavator is shut down. This safety mechanism is a last resort that only kicks in when the controller fails.

In "Manual" mode, the operator drives each of the actuators directly by controlling the rotation speed between the components. In "Assisted" mode, the operator drives the movement of the bucket (the translation in the x and z direction and the angle between the bucket and the ground), while the controller translates the requested motion into the individual rotation speeds of the components. It uses a reverse kinematics calculation based on the measured angles between boom, stick and bucket to generate setpoints for the linear actuators dynamically such that perfectly straight and smooth movements can be made.

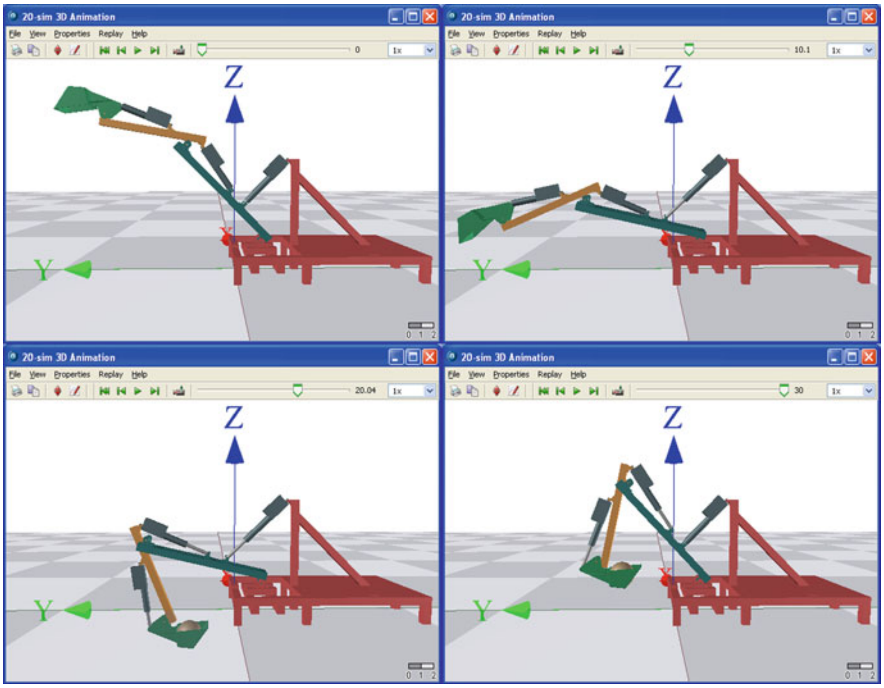

Fig. 11.9 The standard digging motion used in some of the simulations (*from top-left*, *top-right*, *bottom-left to bottom-right*)

11.2.4 Co-simulation Analysis

Using the co-model, different co-simulations have been executed investigating several aspects of the behaviour of the excavator. This information was then used to improve the functionality of the controller and to fine-tune the parameters of the control algorithm. The most interesting results of these co-simulations are presented in this section.

11.2.4.1 Ground Model

In a first experiment, a standard digging motion was performed (see Fig. 11.9) and the mass of the ground load in the bucket was taken into consideration, as well as the load on the actuators. This experiment was repeated for different values of the mass density of the ground, to validate the impact of the ground model on the performance of the system. Results of these simulations are shown in Fig. 11.10. Considering the load of the boom actuator, it is possible that first there is a positive load (bucket goes down), then there is no load (bucket moves towards the actuator) and then there is a

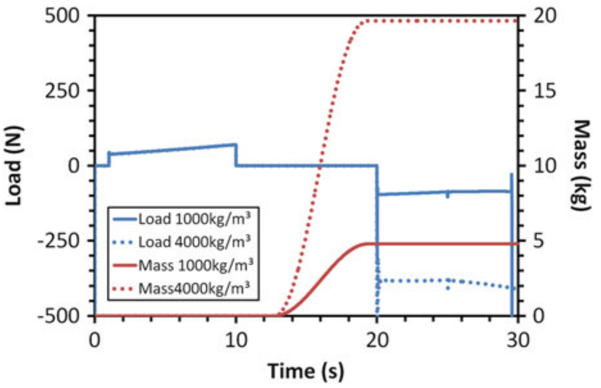

Fig. 11.10 The mass of the load in the bucket and the load on the boom actuator

negative load (bucket goes up). Looking at the graph of the mass, we see that after 13 s, the bucket hits the ground and starts digging, thereby accumulating mass in the bucket. The mass keeps increasing, until the bucket is retrieved from the ground, at 20 s. We can clearly see that the mass in the bucket scales with the mass density of the ground. When the mass density in the bucket is higher, this has a significant effect on the performance of the excavator. This effect only appears when the bucket is pulled up again, thus after it has been filled, which demonstrates that the ground model works appropriately.

11.2.4.2 Overload

One of the aims of this case study is to design a controller that assists the operator so that overloads of the system are avoided. In the controller model, two types of overload protection are included. The first type is protection against abrupt changes to the input of the actuators, resulting in high accelerations and thus loads. An example of this is the situation where the operator pulls the joystick rapidly from one end to the other, resulting in an abrupt switching in the direction of the actuator's motion (i.e. from full speed backward to full speed forward). This abrupt change in input and the resulting load on the actuator are shown in Figs. 11.11 and 11.12. Due to inertia, this results in a very high load on the actuator (the maximum load on the actuator in the test setup is specified at 500 N), enough to cause severe damage to it. To prevent this overload, the controller needs to smooth the joystick input before passing it to the actuator. This smoothing is implemented by a ramp function, defined by a maximum slope (per increase in input voltage per second). A different slope can be set for the forward and backward movement. The figures show the result for an input smoothing with the controller running at 10 Hz and at 100 Hz. Using this smoothing function, we see that the load on the actuator becomes much smaller, thus preventing an overload.

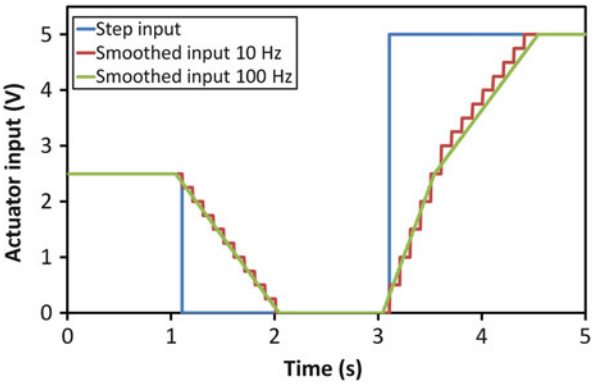

Fig. 11.11 The input signal to the actuator, as function of time. The step input represents the abrupt signal that is received without an intervention of the controller, as opposed to the signals that are smoothed by the controller

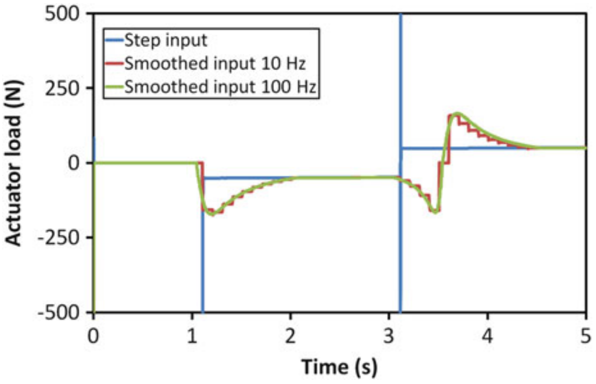

Fig. 11.12 The load on the actuator, for a step and a smoothed input

11.2.4.3 Endstop Protection

The second protection mechanism prevents the actuator from actually crashing into its physical end stops. In order to achieve this, two additional protection mechanisms have been included, one in the controller and one in the safety unit. The input signal and the resulting actuator load are shown in Fig. 11.13. Without any protection mechanism, the input signal to the actuator remains at full speed and the actuator crashes into the end stop, resulting in a very high load.

The protection mechanism of the safety unit terminates motion immediately when such a crash is imminent. Although this mechanism prevents a crash with the end stops, it still does this in an abrupt way which also introduces a high load on the actuator. However, this is acceptable as the hard stop will not be used in every-day operation but only in case of emergency, that is, when the controller fails. In that case, the emergency stop is more important than potential damage to the system, for example a person might be in harms way, which requires immediate action leading to a full stop as soon as possible.

The mechanism modelled in the controller also terminates the motion, but it does this in a smooth way, by slowing down the motion between a soft angular limit and

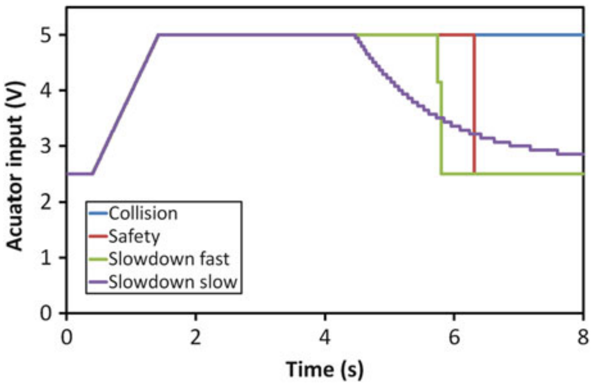

Fig. 11.13 The input signal to the actuator as function of time. The collision signal corresponds to the simulation without any protection, safety refers to the protection mechanism implemented in the safety unit and slowdown is the mechanism implemented in the controller

a hard one. The distance between both limits defines how fast the motion is slowed down, which clearly has an impact on the load on the actuator. When choosing this distance, a compromise has to be made between the maximum allowed load, the performance of the system and the useful angular range of each component. This was determined using the Automated Co-model Analysis feature of the Crescendo technology presented in Sect. 10.2.

11.2.4.4 Emergency Switch

In a final collection of co-simulation experiments, the effect of the emergency switch has been validated. Two co-simulations were performed, where first the system was powered on and then started. Then a certain input signal was given by the joystick and the encoder value of the actuator was logged, as shown in Fig. 11.14. In the second co-simulation, the emergency switch was pressed after 3 s, after which the excavator came to a halt. With the emergency activated, we then tried to restart the system, but the system did not respond, thus indicating that the emergency switch works correctly. Here the system can only be restarted after an entire power-down.

11.2.5 Key Results and Observations

Based on the experiences in the case study, Verhaert concluded that the Crescendo approach has added great value for modelling complex systems like the dredging excavator. They report that it enables teams with different technical backgrounds to work easily together on the same project. Interfaces can be managed by models instead of writing and maintaining lengthy documents (typically tens to several hundreds of pages). The software team can develop and verify the control software in combination with a relevant physical (yet virtual) model. The machine design

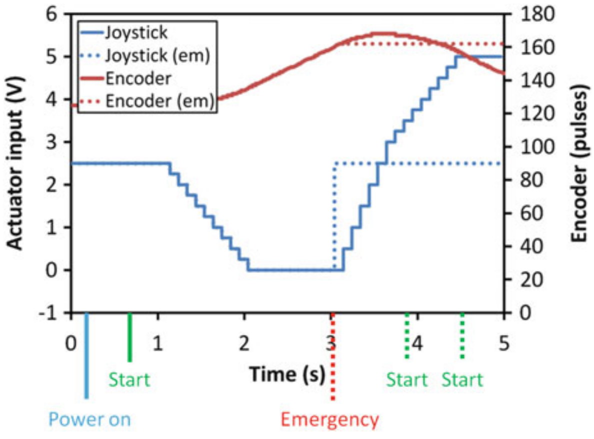

Fig. 11.14 The joystick input and the encoder value as function of time. The different button presses are indicated below the time axis. The *dotted lines* represent the additional button presses in the second co-simulation, the one in which the emergency was activated and the operator tried to restart the system

team can verify the behaviour of the machine as driven with the control software and it can improve the mechanical (or hydraulic) design where needed.

In their opinion, this is particularly interesting for complex controls, for example, multiple states of the excavator and inverse kinematics as part of the control, and for the combination of mechanics and hydraulics. More scenarios than in real life can be tested, and more parameters can be monitored. Our approach also allows for off-nominal or critical simulations that are difficult or dangerous to perform with real hardware. Upscaling the model from a test setup to full scale also allows the designer to check for differences in behaviour between lab scale and full scale. That information can be used to guide the design of the full scale machine and control.

By their detailed modelling of the controller, Verhaert discovered some shortcomings in the initial implementations of the controller that would have been hard to find with other simulation tools. For example while testing the end stops, Verhaert discovered that the assisted mode control behaved in a strange way. After this observation, some changes were made to their controller to accommodate this shortcoming. These changes were first checked in the model and then validated on the setup. Our approach also enabled Verhaert to efficiently validate their emergency supervisor. Last but not least, Verhaert modelled, implemented and successfully demonstrated a fully working version of their "assisted mode" excavator operation on their scale model test setup.

11.3 A Document Handling System

11.3.1 Case Description and Main Challenges

The Neopost case study concerns the model-based design of a document inserting system, whereby different sheets of paper need to be folded and inserted into envelopes. Such a document inserting system, as shown in Fig. 11.15, has a number

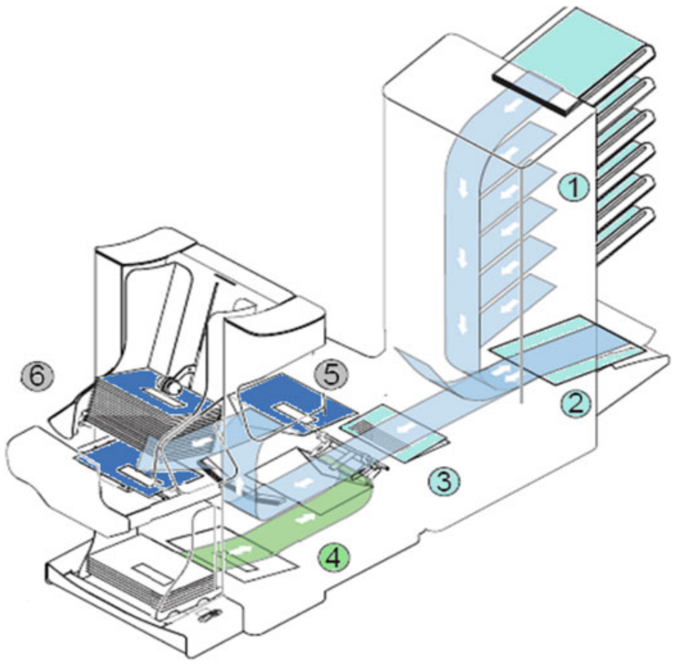

Fig. 11.15 Layout of a Neopost document inserting system: (1) Document bins, (2) collator area, (3) folding area, (4) inserting area, (5) closing area and (6) document exit

of bins containing printed documents and a bin containing empty envelopes. Depending on settings for the application job, the system will separate documents from the available piles (1) in the right order, transport them sequentially to a collator area (2) until the document set is complete. Then, the set will be folded (possibly more than once) and transported to the insert area (3). Meanwhile, an envelope has been separated from the pile of envelopes. This envelope is transported to the insert area and opened mechanically (4). At the insert area, the folded set of documents will be inserted into the envelop (5). Finally, the filled envelope will be glued, closed and transported to the exit of the system (6).

Co-models were developed by Neopost to study the misalignments of documents with respect to each other and / or the heart line of the paper path. This is important, as large misalignments may lead to uncontrolled collisions between documents and envelopes which is critical to quality for document inserting systems, as these collisions may lead to paper jams and hence significant loss of productivity. Understanding how the different causes of misalignment contribute to the total misalignment of a set of folded documents during the insertion process might give direction to how to distribute the available effort over the different possible improvement activities, and this was the main research topic, using our approach.

Due to commercial confidentiality, we cannot show the actual DE and CT models of the document inserting system produced by Neopost. But fortunately, to a large

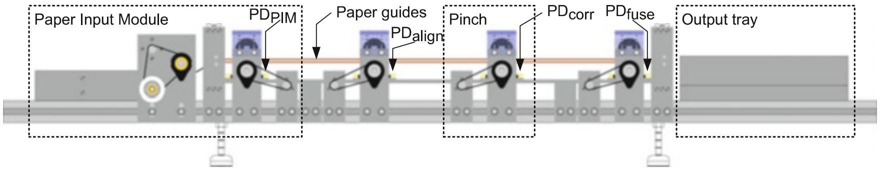

Fig. 11.16 Schematic overview of the experimental setup from [3]

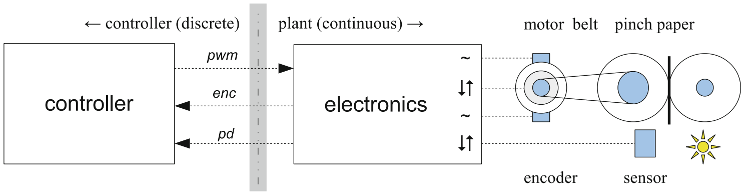

Fig. 11.17 Overview of the controller–plant I/O interface

extent the modelling of this device has great similarity to the paper path control for a digital printer that has been studied in the BODERC project [96]. Thus, we will describe the main characteristics of the BODERC paper path and its experimental setup instead here. Figure 11.16 presents a side view of the paper path. It consists, from left to right, of a single paper input module with a built-in sheet separation device, and four pinches to transport the paper towards the output tray. Each pinch is driven by an electric motor through a belt and each motor is equipped with a rotary encoder. Simple paper guides are used to lead the paper from pinch to pinch and optical sensors are available to detect the edges of the sheet.

Figure 11.17 presents the overall controller architecture, for a single motor–pinch pair. The digital controller produces a PWM signal which is connected to a power amplifier. The power amplifier produces an electric current that is proportional to the PWM signal. The motor torque is in turn proportional to the current supplied by the amplifier. The motor torque is transferred to the pinch through a belt, while rotation of the motor axle (and *not* the pinch axle) is measured by a rotary encoder. Each pinch consists of two rubber rollers which are mounted such that their surfaces touch. Paper can then be transported due to friction by inserting a sheet in between the rubber rollers. The position of the leading and trailing edge of a sheet can be measured with an optical sensor, the paper detector, which triggers as soon as the light levels are changed by the passing sheet.

The paper path setup emulates the behaviour of a high-volume digital printer, whereby each motor–pinch pair has a specific control function to perform. The first pinch is used to stabilise the speed of the paper coming from the paper input module. The second pinch is used to decelerate, stop and accelerate each sheet in order to simulate the alignment process. The purpose of the third pinch is to deliver the sheet at exactly the right time and at the right constant speed at the fourth pinch. The fourth and final pinch represents the printing process, where the image is put on the sheet. This pinch delivers each sheet in the output tray. The overall control goal is to maximise throughput while ensuring that individual sheets never touch each other while in transit, and ensuring that alignment of each image on each sheet remains within the set accuracy requirement. We will first introduce the continuous time model in Sect. 11.3.2 and then the discrete event model in Sect. 11.3.3. Afterwards Sect. 11.3.4 presents the co-simulation analysis conducted on this co-model and finally Sect. 11.3.5 provides an overview of the key results of this study.

11.3.2 The Continuous Time Model

The top-level bond graph model of the plant is shown in Fig. 11.18. At the bottom of the figure, we see the interface towards the controller. There is a pair of PWM and encoder signals connected to each motor-belt-pinch icon.[1] These icons represent lower level bond graph models, or submodels, which we will present later in more detail. The plant model has four motor-belt-pinch submodels while our experimental setup has five. The first motor in the setup is only used to inject new sheets into the paper path. Since its operation is only of minor importance to the total system behaviour it is only abstractly represented in the plant model by means of the `FeedSheet` signal. The behaviour of the individual sheets is represented by the photographic icon. This submodel maintains the state of each sheet in the paper path, such as for example its current speed and position. The state of the `PaperDetectors` signal is automatically derived from this information. If the position of a paper detector is in between the leading and trailing edge of at least one sheet then it will yield 1 else 0.[2] Similarly, the sheet is under the control of a pinch if the position of the pinch is in between the leading and trailing edge of the current sheet position. The animation icon is used as a monitor which allows us to visualise the simulation graphically.

The pinches drive the sheets and this transfer of energy is influenced by friction. In kinematic models the friction is assumed to be zero but this is usually not very realistic. The friction force which is imposed on each sheet is a function of the mass of the sheet and the speed difference between the sheet and pinch. Of course, the friction force is imposed if and only if the sheet is in control of a pinch. What happens if a sheet is under the control of two pinches? In our model, we assume that the pinch near the leading edge of the sheet dominates the pinch near the trailing edge. The assumption is that the force imposed by the leading edge pinch will cause the sheet to slip in the trailing edge pinch. This abstraction can be used if and only if the speed of the leading edge pinch is equal to or slightly higher than the speed of the trailing edge pinch. This condition can be checked at simulation time. Only a very trivial friction model is used here, but it can simply be replaced by more complex hybrid friction models if the need arises, without affecting the plant model architecture demonstrated here.

[1]The Neopost document handling system also makes use of pinches for transporting the paper.

[2]Similar paper detectors are used in the Neopost document handling system.

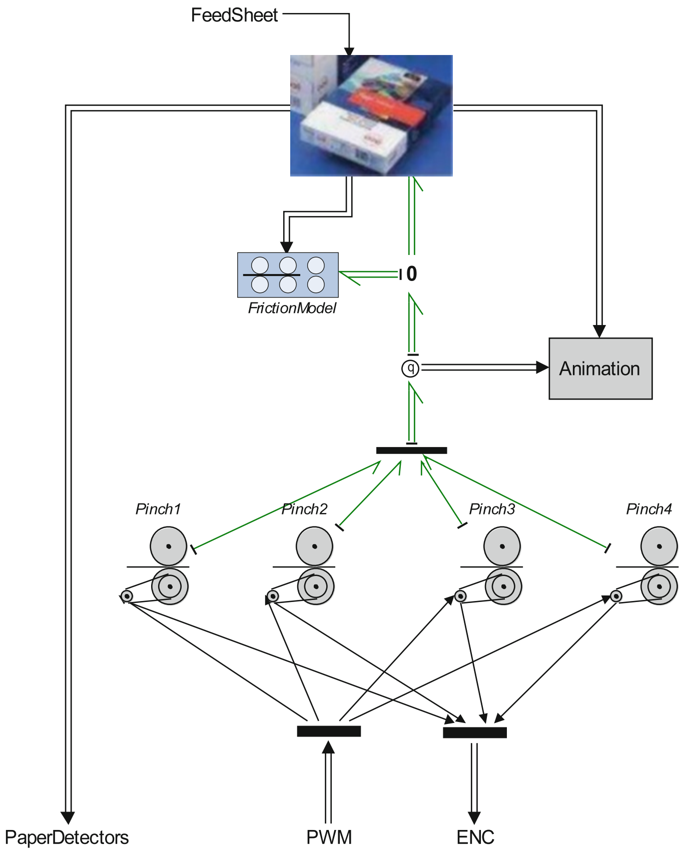

Fig. 11.18 Top-level bond graph of the plant model

The bond graph submodel for the motor, belt and pinch is presented in Fig. 11.19. This iconised diagram demonstrates at a very high level of abstraction how the control signal relates to the movement of the sheet. For example, observe that the behaviour of the power electronics, the so-called H-bridge, has been modelled as a simple multiplication (or gain) factor. In other words, the amount of power provided to the motor is linearly proportional to the duty cycle of the PWM input signal obtained from the controller. The motor converts this electrical energy into torque. The torque causes the belt to rotate and the belt in turn drives the pinch, whereby the “Belt and Gear” submodel simply multiplies the rotational speed of the motor by the gear ratio. And finally, the pinch transfers its energy towards the sheet of paper as described previously. The angular velocity is measured at the motor axis and this value is multiplied by 2π to obtain the number of rotations per second. We will see later how this value is converted into encoder values in the detailed I/O interface model.

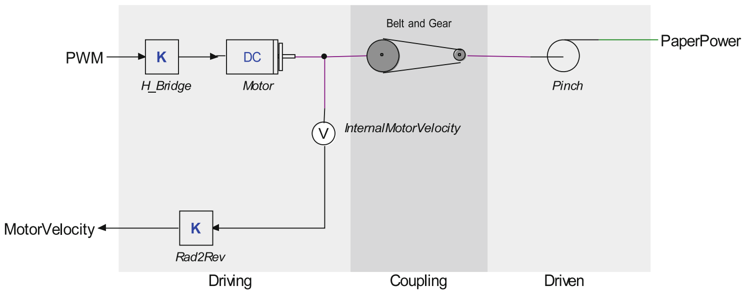

Fig. 11.19 Bond graph of the motor, belt and pinch subsystem

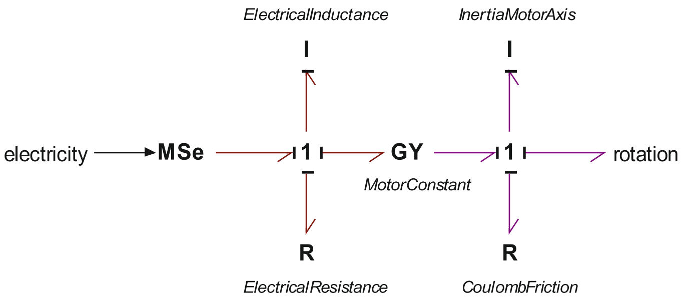

Fig. 11.20 Bond graph of the DC motor from Fig. 11.19

The "DC motor" icon in Fig. 11.19 is itself a bond graph submodel as shown in Fig. 11.20, which again demonstrates the explicit hierarchy in the model. The inductance L, internal resistance R, motor torque constant, rotor inertia and Coulomb friction parameters required by this submodel can usually be found in the supplier data sheet.

Finally, we revisit the I/O interface model. The I/O model acts as a mediator between the discrete time controller model and the continuous time plant model, as shown earlier in Fig. 11.17. In other words, discrete values need to be converted into continuous signals and vice versa and we have to ensure that this conversion process does not affect the overall analysis at the system level. A detailed overview of the I/O model is shown in Fig. 11.21. The top row demonstrates from left to right how the discrete PWM values are converted into their continuous counterpart, in this case an analog voltage between $\langle -1, 1 \rangle$ V.

11.3.3 The Discrete Event Model

The control application will have to satisfy different goals simultaneously. The behaviour of a sheet, in terms of its speed through the paper path, is graphically presented in Fig. 11.22. The numbers 2–4 correspond to the pinch that is in control of the sheet at a given point in time. The grey areas indicate where the paper is in control of two pinches simultaneously.

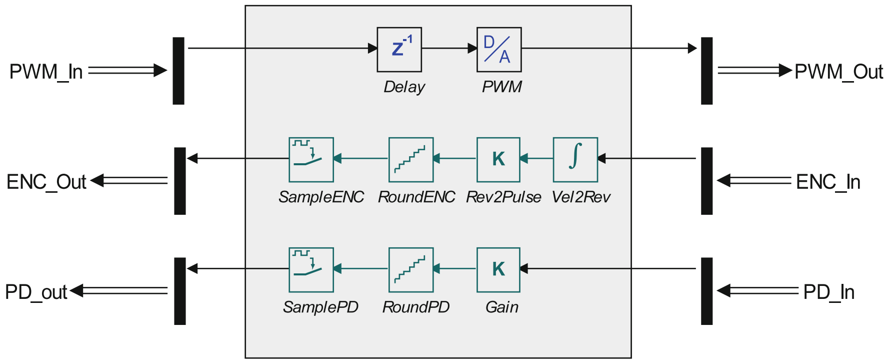

Fig. 11.21 Detailed overview of the I/O interface model

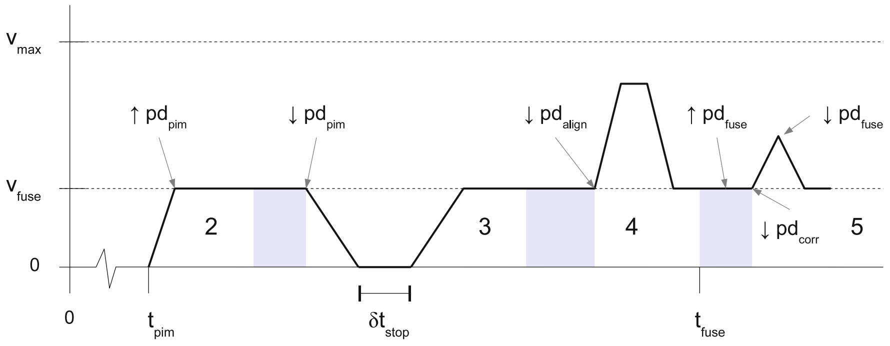

Fig. 11.22 Overview of a typical sheet velocity profile

This velocity ω is determined by the required system throughput performance as described in the following equations:

$$V_{\text{nom}} = (pagesize + isd) \cdot tp/60 \qquad (\text{mm/s}) \tag{11.1}$$

$$\omega = V_{\text{nom}}/2\pi \cdot r_{\text{pinch}} \qquad (\text{rad/s}) \tag{11.2}$$

whereby *pagesize* represents the length of a sheet (in mm), *isd* represents the inter-sheet distance (in mm), *tp* represents the throughput (in pages per minute) and finally r_{pinch} represents the radius of the pinch (in mm). The inter-sheet distance is defined as the distance between the trailing and leading edge of two consecutive sheets. The primary task of the paper path subsystem is to deliver each sheet on time and at the right speed at the pinch.[3] This requirement has two implications:

[3]For the Neopost document handling system, similar requirements are present for preventing skewing of paper when folding.

1. With respect to *on time*. The inter-sheet distance shall be maintained in order to meet the required system performance and to ensure the correct alignment of the image on the sheet. A maximum deviation of 0.5 mm is allowed exactly at the pinch but the inter-sheet distance may vary elsewhere as long as two consecutive sheets do not collide or overlap.
2. With respect to *right speed*. The leading edge of each sheet shall have the nominal speed V_{nom} just before it is in control of the pinch. A maximum deviation of V_{nom} of 2 % is allowed.

Now the main control goal has been identified, we can look at the secondary tasks to perform by the control application. Considering the pinches in the experimental setup, we have the following additional requirements:

1. The first pinch is part of the paper input module and it is used to retrieve sheets from the tray. The challenge is to ensure that single sheets are separated. The solution is to control this motor belt pinch subsystem in open loop. Basically the motor is told to accelerate as fast as possible for a very short period of time and then immediately decelerate. The friction force between the pinch and the top sheet is larger than the friction force between the top two sheets, which will cause clear separation of a single sheet.
2. The purpose of the second pinch is to get the sheet under control by moving it down the paper path at the nominal speed V_{nom}.
3. The purpose of the third pinch is to decelerate, stop and accelerate the sheet. The length of the stop period can be defined by the super-user setting up experiments on the paper-path.
4. The purpose of the fourth pinch is to ensure that the sheet is delivered with the correct inter-sheet distance and speed to the exit of the paper path. It will have to compensate for the time lost during alignment of the sheet at the previous pinch.

The control application will have to satisfy all these sub-goals simultaneously. The behaviour of a sheet, in terms of its speed through the paper path, is graphically presented in Fig. 11.22. The numbers 2–4 correspond to the pinch that is in control of the sheet at a given point in time. The grey areas indicate where the paper is in control of two pinches simultaneously.

We will take a step-by-step look at the lifetime of a sheet during its travel through the pinches in order to get a feeling for the control complexity involved. The events mentioned are also shown in Fig. 11.22 (similar to the DE-first approximations shown in Fig. 8.16b).

1. A new print job arrives and the image processing starts. Meanwhile, pinches 2–4 are booted up until they run at V_{nom} and then the first sheet is requested from the paper input tray. The sheet is separated by pinch 1 and it is inserted into the paper path. It will hit pinch 2 with some force and at the wrong speed since we use a rather brute force separation method.
2. Pinch 2 accepts the first sheet and will try to stabilise its speed to V_{nom}. The timing tolerance caused by the brute force separation is known when the leading edge of the sheet is detected by $\mathtt{PD}_{pim}$, as shown by the $\uparrow$-arrow.

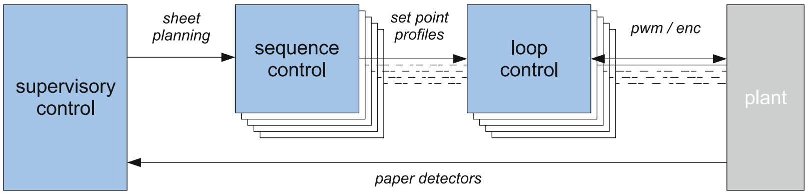

Fig. 11.23 An informal overview of the three tier control application architecture

3. The sheet continues to move downstream and gets into joint control of pinches 2 and 3 (the first grey area in the figure). The control application knows that the sheet has left the control of pinch 2 when the trailing edge of the sheet is detected by $\mathtt{PD}_{pim}$, as shown by the $\downarrow$-arrow. The alignment phase of the sheet can now start because pinch 3 is in full control. The deceleration needs to be quick enough to ensure that the leading edge of the sheet does not reach pinch 4.
4. The sheet is accelerated to V_{nom} after the user-defined alignment time $\delta\mathtt{t}_{stop}$ has expired. The acceleration must be performed quickly to ensure that the nominal speed has been reached before the leading edge of the sheet hits pinch 4.
5. The sheet continues to move downstream and gets into joint control of pinches 3 and 4 (the second grey area in the figure). The control application knows that the sheet has left control of pinch 3 when the trailing edge of the sheet is detected by $\mathtt{PD}_{align}$, as shown by the $\downarrow$-arrow. The correction phase of the sheet can now start because pinch 4 is in full control. The sheet is accelerated to compensate for the time lost in the alignment phase and the tolerances caused by the brute force sheet separation.

The informal description of the requirements for the control application listed above gives us some inspiration for the control application architecture that is required to address these challenges. From the engineering point of view, it is usually a good idea to apply the “separation of concerns” principle, for example to divide the time critical parts from the less time critical parts of the application. In fact, we will provide each motor-belt-pinch subsystem its own controller in our architecture because the requirements differ and they may be deployed on different hardware in the final implementation. In contrast, the timing requirements for the high-level sheet flow control, as presented in the sheet life-cycle are far less demanding and a single application may suffice for this purpose. The linking pin between this high-level supervisory control layer and the real-time controllers is a set of so-called sequence controllers, one per real-time controller. These sequence controllers generate so-called setpoint profiles ahead of time, based on the planning information received from the supervisor. An informal overview of this well-known three tier control application architecture is shown in Fig. 11.23.

An extract of the VDM-RT models for the controller application architecture is presented in a bottom-up fashion. We start at the plant model interface and the real-time loop controller and work our way up towards the supervisory control. Each motor-belt-pinch subsystem has an interface consisting of a PWM input and encoder output (ENC). This interface is well suited for feedback control. A standard PID control strategy will be used for pinches 2–5. The real-time controller will periodically sample the encoder value. This value is a measure for the distance covered and it is compared to the so-called setpoint, which represents the intended value. The difference between the two is called the error and with the PID algorithm we calculate a new PWM value to compensate for this measured error, by accelerating or decelerating the motor accordingly. The four PID loop controllers will operate at 1 kHz in our controller models. Open loop control is used for pinch 1. Basically, the setpoint is forced upon the motor-belt-pinch system by writing the correct PWM value but the encoder value is ignored. For convenience, the loop controller for pinch 1 will also run at 1 kHz.

11.3.3.1 The Loop Controller

Consider the VDM-RT model for the loop controller shown below. The constructor of the active class `LoopController` takes two arguments. The first argument, `ptp`, determines whether the calculated output value is sent to the plant model at the start of the next iteration or immediately. The second argument, `pfb`, is used to distinguish the control strategy: closed loop or open loop.

```
class LoopController

instance variables
  -- time-triggered (true) or immediate output (false)
  private hold : bool := true;
  -- closed loop (true) or open loop (false)
  private feedback : bool := true

operations
  public LoopController: bool * bool ==> LoopController
  LoopController (ptp, pfb) ==
  ( hold := ptp; feedback := pfb )
```

The instance variable `output` is used to temporarily store the calculated PWM value. The operation `CtrlLoop` implements the real-time control strategy and this operation is periodically executed.

The operation `getSetpoint` used inside `CtrlLoop` retrieves the setpoint from the passive `SetpointProfile` object for the current local time `ltime`. Setpoint profiles are the interface between the loop and sequence controllers. The setpoint profile can be updated by the sequence controller by calling the asynchronous `addProfileElement` operation. The operations `getSetpoint` and

```
instance variables
  private output : real := 0;
  private ltime : real := 0

operations
  public CtrlLoop: () ==> ()
  CtrlLoop () ==
    -- first retrieve the current encoder value
    ( dcl enc : real := getEnc();
      -- update the old output if time-triggered
      if hold then setPwm(output);
      -- calculate the new PWM value
      output := if feedback
                then limit(calcPID(enc))    -- closed loop
                else limit(getSetpoint()); -- open loop
      -- update the output if not time-triggered
      if not hold then setPwm(output) )

thread
  -- execute the controller at 1 kHz
  periodic (10E6, 0, 0, 0)(CtrlLoop)
```

addProfileElement are declared mutual exclusive to prevent data corruption by simultaneous access to the profile instance variable.

```
instance variables
  profile : SetpointProfile := new SetpointProfile()

operations
  private getSetpoint: () ==> real
  getSetpoint () == profile.getSetpoint(ltime);

  async public
  addProfileElement: real * real * real ==> ()
  addProfileElement (px, py, pdt) ==
    profile.addElement(px, py, pdt)

sync
  -- access to the profile is mutual exclusive
  mutex (addProfileElement, getSetpoint);
  mutex (addProfileElement)

end LoopController
```

11.3.3.2 The Setpoint Profile

The passive class `SetpointProfile` is used as a container to collect all knowledge about manipulating so-called setpoint profiles. A setpoint profile is an ordered collection (a sequence) of left-closed, right-opened, line elements which together define the evolution of the setpoint over time similar to the strategy followed in Sect. 8.5.4. Each line element, or `ProfileElement`, is defined by three real numbers. The first number defines the domain: the starting time t at which this element is valid. The second and third number define the range: the current value at time t and the direction coefficient that is valid from this point in time onwards respectively. Setpoint profiles are defined from some point in time to infinity, since the last element in the profile is right-opened. The invariant of the `profile` instance variables ensures that the domain is strictly monotonically increasing but it does allow discontinuities in the range. The operation `addElement` can be used to extend the current setpoint profile.

```
class SetpointProfile

types
  private ProfileElement = seq of real
  inv pe == len pe = 3

instance variables
  profile : seq of ProfileElement := [];
  inv forall i, j in set inds profile &
        i < j => profile(i)(1) < profile(j)(1)

operations
  public addElement: real * real * real ==> ()
  addElement (t,v,a) ==
    profile := profile ^ [[t,v,a]]
  pre len profile > 0 => profile(len profile)(1) < t
```

The operation `getSetpoint` is used to compute the actual setpoint at some specific point in time based on linear interpolation of the abstract continuous time description maintained in the `profile` instance variable. Consider the example

```
[ [0,0,0], [1,0,1], [2,1,0], [4,1,-1], [5,0,0]]
```

This trapezoid setpoint profile ramps up from $\langle 1,2\rangle$ and ramps down from $\langle 4,5\rangle$. Hence, `getSetpoint(1.5)` would yield the value 0.5.

```
operations
  public getSetpoint: real ==> real
  getSetpoint (t) ==
    if len profile = 0
    then return 0
    else ( dcl prev_pe : ProfileElement := hd profile;
           for curr_pe in tl profile do
             if curr_pe(1) > t
             then return calcSetpoint(t, prev_pe)
             else prev_pe := curr_pe;
           return calcSetpoint(t, prev_pe) )
  pre t >= 0 and len profile > 0 => t > profile(1)(1)

functions
  private calcSetpoint: real * ProfileElement -> real
  calcSetpoint(t, [px, py, pdydx]) ==
    py + pdydx * (t - px)
  pre t >= px

end SetpointProfile
```

11.3.3.3 The Sequence Controller

The active class `SequenceController` contains the knowledge to translate high-level paper path planning commands into setpoint profiles that are used by the loop controllers. Each sequence controller is associated with exactly one loop controller `loopctrl`.

```
class SequenceController

instance variables
  public loopctrl : [LoopController] := nil
```

The operation `initNominal` is used to power-up the pinches until they reach the nominal paper path speed `v_nom`. The motors are not started at full throttle immediately, but they are ramped up gradually. The user can influence the power-up time by setting the acceleration parameter `a_nom`.

The operation `setStopProfile` is used to bring the sheet in the paper path to a complete stand still for `dstop` seconds. The procedure will start at t_1 with speed v_1 mm/s and the sheet will accelerate and decelerate with `acc` mm/s^2.

```
operations
  async public initNominal: real * real ==> ()
  initNominal (v_nom, a_nom) ==
    ( -- ramp up the motor to the nominal paper speed
      loopctrl.addProfileElement(0, 0, a_nom);
      -- and maintain a constant speed indefinitely
      loopctrl.addProfileElement(v_nom/a_nom, v_nom, 0))
    pre v_nom > 0 and a_nom > 0 and loopctrl <> nil;

  async public initPeak: real ==> ()
  initPeak (tpeak) ==
    -- give the sheet a good kick for 60 msec
    ( loopctrl.addProfileElement(tpeak, -40, 0);
      loopctrl.addProfileElement(tpeak+0.060,0,0) )
    pre loopctrl <> nil
```

```
operations
  async public
  setStopProfile: real * real * real * real ==> ()
  setStopProfile (t1, v1, acc, dstop) ==
    def dt = v1 / acc in
      ( loopctrl.addProfileElement(t1, v1, -acc);
        loopctrl.addProfileElement(t1+dt, 0, 0);
        loopctrl.addProfileElement(t1+dt+dstop, 0, acc);
        loopctrl.addProfileElement(t1+dt+dstop+dt, v1,0))
    pre acc <> 0 and loopctrl <> nil

end SequenceController
```

11.3.3.4 The Supervisory Controller

The active class `Supervisor` represents the supervisory control in our architecture. It has four instance variables of type `SequenceController`. The links to these objects are created at model instantiation time. Each sequence controller takes care of one motor-belt-pinch subsystem.

The core functionality of the supervisory control application is captured in the operations that respond to the paper detectors. For example, the `pimDownEvent` operation will be called whenever a trailing edge of a sheet has been detected by paper detector $\texttt{PD}_{pim}$. This event signals the start of the alignment process which will bring the sheet to a complete stand still, in our case for 100 ms.

```
class Supervisor

instance variables
  public ejectSeqCtrl : [SequenceController] := nil;
  public pimSeqCtrl   : [SequenceController] := nil;
  public alignSeqCtrl : [SequenceController] := nil;
  public corrSeqCtrl  : [SequenceController] := nil;
...
```

```
operations
  -- operation to initiate the alignment procedure
  async public pimDownEvent: () ==> ()
  pimDownEvent () ==
    -- start decelerating in 10 msec from now
    def dectime = time + 0.01 in
      alignSeqCtrl.setStopProfile(dectime, 50, 500, 0.1)
    pre alignSeqCtrl <> nil
```

11.3.4 Co-simulation Analysis

In the case of the paper path models presented in the previous section, we encountered two significant problems that were only exposed during co-simulation. Interestingly, neither problem was found during CT-only analysis nor during DE-only analysis, which demonstrates the added value of our approach.

The first problem was a simple mistake with potentially severe consequences. The VDM-RT controller model used mm/s as the unit of measure in the setpoint profiles, but the loop controller measured the distance covered in radians. Hence, the integrated setpoint values provided to the PID controller were incorrect, causing the wrong output values to be calculated because the error was off the chart at every iteration, leading to the constant spin-up of the motor at maximum speed. The root cause of this problem was easily identified since it is very simple to monitor model parameters during simulation. It would have taken substantially more time and effort if the cause of the problem had to be investigated on the embedded target (implementation).

The second problem was slightly more complex but is also due to a misinterpretation of the informal requirements. The designers of the plant model assumed that the time earmarked for the alignment of the sheet also included the time required to decelerate the paper. However, the designers of the controller model followed a strict interpretation of the requirement: the time needed to decelerate is not included in the alignment time. The designers of the plant model performed a simulation using a simplified controller model and claimed that an inter-sheet distance of 50 mm was feasible at a productivity rate of 50 pages per minute and an alignment time of

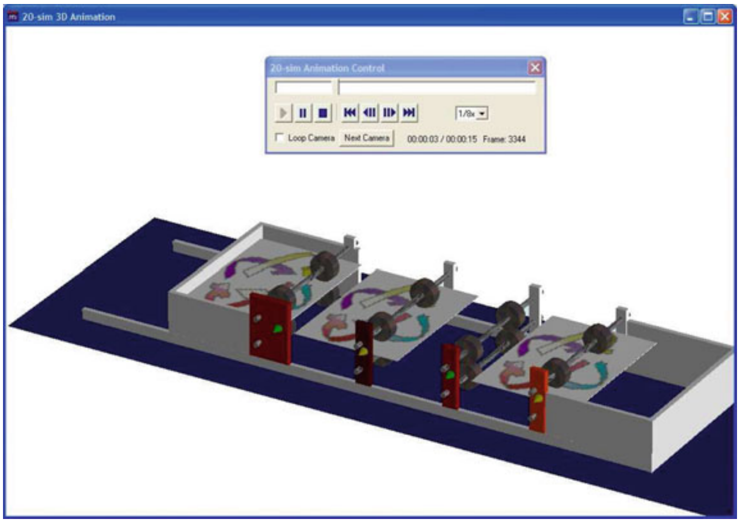

Fig. 11.24 3D visualisation of the paper path co-simulation in 20-sim

200 ms. However, when the experiment was performed on the experimental setup it turned out that their controller model was incorrect and they circumvented the problem by increasing the inter-sheet distance to 100 mm, reduced the alignment time to 100 ms and operated the alignment motor with maximum acceleration and deceleration values.

As sometimes happens in real life, these lessons learnt were not properly documented and communicated. Therefore, the designers of the controller model based their design on the wrong data. This issue became very clear when the controller model was tested in combination with the plant model. Both the different assumption about the same requirement as well as the lack of communication of the insight gained from the plant experiments were easily identified using the visualisation capabilities of 20-sim. The plant model designers developed a three dimensional model of the paper path as a plug-in to their plant model. This interface is shown in Fig. 11.18. The visualisation runs in parallel with the co-simulation, whereby the virtual prototype is fully synchronised with the simulation state. It also provides the ability to stop, rewind and replay the visualisation such that the system behaviour can be inspected in detail. Using this facility it was demonstrated convincingly that two consecutive sheets would always collide if the original parameters were used. An example of the visualisation is shown in Fig. 11.24.

Both examples illustrate a key problem occurring in industrial practice: errors are made in critical design parameters (unvalidated assumptions) due to lack of communication in the design team. Co-simulation exposes these issues early in the development life-cycle.

11.3.5 Key Results and Observations

Neopost Technologies confirms these findings from the BODERC project as they had similar experiences as those listed in the paper path case study. The co-modelling approach forced the CT- and DE-groups of engineers to communicate right from the start of the project, which was experienced as being of great value for the mutual understanding and overall project progress.

With respect to the system under development, Neopost identified and solved several flaws in the system design due to the co-modelling work performed. Actually, these defects were discovered even before physical parts of the system were realised, which saved several weeks of development time. In fact, the entire software implementation was written and tested using a system mock-up that was directly based on the Crescendo co-simulation models. This allowed parallel development of both hardware and software and also significantly reduced the time required to perform software integration when the physical system became available. This integration process was also smoother than usual, saving again several man months in development lead time. But even when the hardware became available, the Neopost engineers kept using and maintaining the development approach based on the Crescendo models, because of the following:

- In the early stages of the project, the simulator was working more reliably and was more flexible than the real physical system. This meant that by using the simulator instead of the real machine, the engineers could concentrate on the development of their software instead of keeping the mechanics running.
- The availability of the simulator was greater than the availability of the real machine. The system prototype turned out to be under constant development and near constant maintenance led to low availability for the software engineers. This became of crucial importance when the development team grew and only a few systems where available.

11.4 The ChessWay Self-balancing Scooter

11.4.1 Case Description and Main Challenges

The ChessWay case study and its main control challenges were introduced in Sect. 7.3, and models of the ChessWay have been used to illustrate the DE-first approach to co-model creation (Sect. 8.5.4), and approaches to the modelling of faults and fault tolerance (Sect. 9.8). In this section, we report on the actual development of the ChessWay study within Chess.

One of the main motivations for the study was the need to explore the range of potential faults. In fact, before following the DE-first approach presented in Sect. 8.5.4, an abstract model was produced of the full system with the purpose of identifying interfaces, considering safety aspects and analysing potential fault handling strategies as early as possible. The emphasis of this model has been on initial exploration of the system complexity, rather than on model structuring and analysis from a traditional object-oriented point of view, or co-simulation. However, this model provided valuable insights that have been transferred over into the DE-first approach shown in Sect. 8.5.4. An overview of this initial model can be found in Appendix D.

11.4.2 The Continuous Time Model

The initial CT model described the dynamic behaviour of the ChessWay self-balancing scooter limited to two-dimensional space, so it could only ride forward and backwards, and assumed only a single-driven wheel, whereby the digital (closed loop) control was embedded deep within the CT model. When co-simulation with the initial DE model was acceptable, the CT model was expanded to a 3D setting, with two independent driven wheels including steering, and potential faults were also incorporated. Moreover, the closed loop control was partly moved from the CT model to the DE model. This top-level CT model is presented in Fig. 11.25. At the top, we see two independent PWM signals, coming from the DE model, that drive the power amplifiers, one for each wheel. Each wheel has an explicit contact model which reflects the energy transfer between the wheel and surface, which in turn might depend on its actual location. This allows the execution of experiments with different surface models, independent from the system model. Last but not least, the CT model provides the current forward speed and the angle of the handlebar as input back to the DE model.

11.4.3 The Discrete Event Model

A distributed controller architecture was chosen for the ChessWay, whereby each wheel has its own controller. This was done on purpose in order to demonstrate and expose typical reliability challenges that occur in these type of architectures. Each motor controller is guarded by a *safety*, which has the task to intervene and put the system in a fail-safe state if a fault is detected. The fail-safe state condition for the ChessWay is the situation whereby both motors are not actuated and free running. Furthermore, the system should be able to recuperate from such an intervention and return to normal operating mode if the root cause of the fault has been removed, for example, due to the user's intervention.

An overview of the distributed controller architecture of the ChessWay is shown in Fig. 11.26. Each controller has its own safety monitor and a motor controller. The architecture seems mirrored, but note that only one of the safety monitors has direct physical access to the safety key, the other safety monitor needs to communicate over the bus connecting the two controllers to access the device remotely, see Fig. 7.5. If the safety key is removed, then the safety monitor will move to the

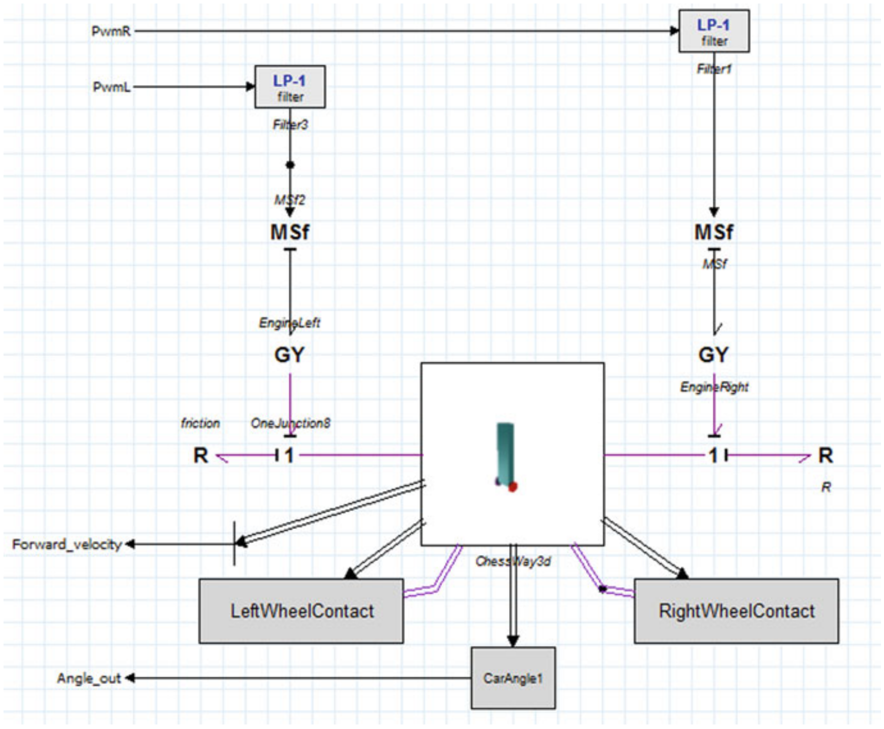

Fig. 11.25 Top-level CT model of the ChessWay

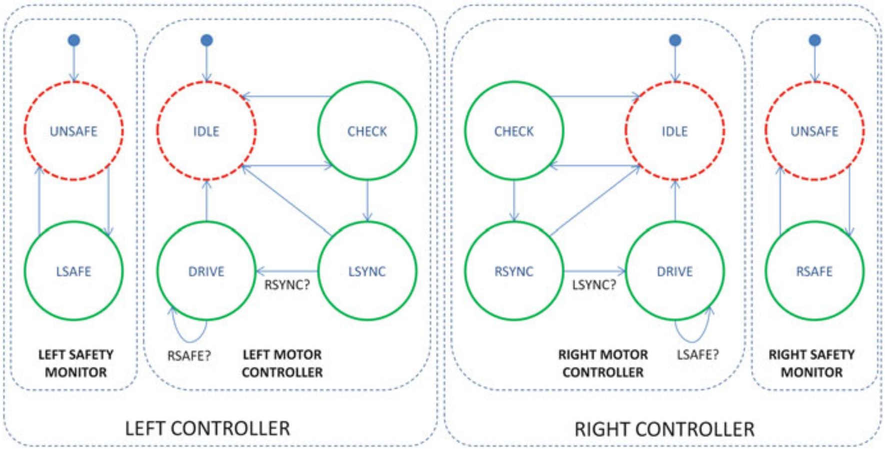

Fig. 11.26 Distributed DE controller architecture

UNSAFE state and the wheel will be undriven. In that case, the safety monitor will force the motor controller to go to the IDLE state. Note that the safety monitor only affects the local motor controller so some synchronisation is required with the other controller. If the safety key is inserted then the motor controller will move from IDLE to CHECK mode. The motor controller will move from CHECK to SYNC mode, whenever the ChessWay has been kept upright for at least 3 s (this test is not shown here). This is to prevent the user being hit by the handle bar when the ChessWay is lying flat when the power is turned on. The motor controller will move from SYNC to DRIVE mode if and only if the other motor controller is also in the SYNC state. The general idea being that both motor controllers will proceed to DRIVE mode at the same time. The motor controller will execute the control algorithm to keep the ChessWay upright and steer only while in DRIVE mode. It will continue to remain in DRIVE mode for as long as the safety monitors on both controllers are in the SAFE state. Again, note that the motor controller needs to communicate to its neighbour to inquire its current safety state. This implies that the communication path between the left and right controller potentially affects the system reliability, as communication may corrupt or delay data, or even fail entirely.

11.4.4 Co-simulation Analysis

Results of a co-simulation are shown in Fig. 11.27, with, from top to bottom:

- The pole angle (in radians).
- The state of the safety switch (1 = CLOSED, 0 = OPEN).
- *PWMInR*, the right-hand-side PWM value, being the equivalent to the average current (in A) to the motor. This should be the same as the following curve, multiplied with the safety switch. That means that as soon as the safety switch is opened by the safety monitor, the *PWMInR* will become zero, which is indeed the case.
- *PwmSettingR* the right-hand-side PWM setting as it is transferred from DE to CT. This signal continues until the state machine of the right-hand-side controller has changed its state from DRIVE to CHECK. This is shown in the graph.
- Status of the right-hand-side controller (0 = IDLE, 1 = CHECK, 2 = SYNC, 3 = DRIVE).
- Status of the right-hand-side safety monitor. (0 = UNSAFE, 1 = SAFE).

We show here a specific simulation to demonstrate the cooperation between the DE and CT part of the co-simulation. It shows the closed loop control, stabilising the vehicle and handling of a simulated error. The vehicle starts in an off-balance position of 0.1 rad in the controller state DRIVE and safety state SAFE. The real-time controller generates a PWM signal to bring the pole angle to zero rad. Initially the PWM signal peaks to generate a relatively large current, to force the pole angle towards zero to prevent the vehicle from falling. At 0.205 s, a fault was introduced by pulling the safety key. Immediately the safety monitor changes from SAFE to

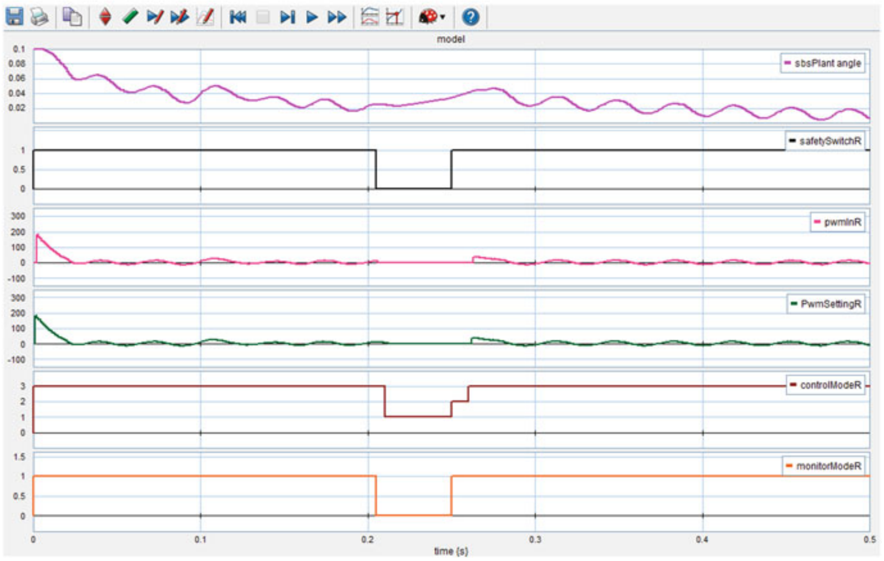

Fig. 11.27 Simulation result: (1) the controller forces the vehicle to change from 0.1 rad approaching zero, (2) handling of an error that is introduced at 0.205 s and (3) recovery of that error from 0.25 s onwards

UNSAFE and the safety switch changes from CLOSED to OPEN. The PWM signal (average current to the motor) should become zero here as well, to simulate open loop wires to the motor (safety switch open). The control status changes to CHECK as well, but a little later. This is to show that the emergency fault handling acts faster and therefore overrules the slower running controller. The safety key is inserted again at 0.25 s, which causes the safety monitor and safety switch change to SAFE and CLOSED, respectively. The controller detects the safe state and changes to SYNC and after both sides are in SYNC, it changes state to DRIVE again after which the motor is powered by a PWM signal.

The diagram nicely shows the relationship between the dynamic behaviour of the vehicle and the state changes of the controller, monitor and safety switch. It also clearly demonstrates the purpose of having the graphical presentation of the DE states in the same diagram as the CT graphs.

11.4.5 *Key Results and Observations*

The DE model helped the stakeholders to differentiate precisely the real-world phenomena that take place and their interpretation (observation) in the control software. Originally, a fault was injected in the DE model, and therefore directly seen by the DE model and reacted to instantaneously. However, in real products, there is always a delay between the occurrence of a fault to its detection by sensors and followed by the response of the control software. This leads to a difference between the software's picture of the physical system, and the physical system's real state.

It is important to take account of this delay because too large a detection time can cause hazards to users. This distinction has been modelled by the explicit definition of distinct plant (real world) and software states. The software interacts with the plant via sensors and actuators; errors and faults in the plant are discovered by the software only via sensors, just as in real systems. The software transforms one state instance to the other. Several processes that handle such actions have been identified, for example

- An actuator representing the PID controlled pulse with modulator of the motor current.
- A controller, being a state machine for discrete control, such as startup, shut down, handling of faults.
- A safety monitor to detect errors and perform an emergency reaction independent from the controller.

The DE modelling has helped to identify these independent processes early and they have been maintained throughout the evolution of the models, leading to a very high correspondence between the abstract initial models and the elaborated models used for co-simulation. This has added to the confidence in the model correctness and likewise for the (manual) implementation that followed. The only errors that were found in the implementation had to do with the "glue code" between the computing hardware and the handwritten implementation of the model. As the structure of the implementation followed the structure of the co-simulation models, it was relatively easy to spot and fix the mistakes made, which reduced the test- and integration effort significantly.

The ChessWay case study contributed a great deal to identifying the *conceptual* boundaries between the DE and CT models, as reflected in Chap. 8, including the insight that the interface between them is actually a layer in its own right, as shown in Figs. 3.40 and 11.21. The co-simulation contract basically reflects an arbitrary "knife-cut" boundary drawn right through this I/O layer. Consequently, some elements of the interface end up as part of the DE model while the rest becomes part of the CT model. This not only affects the way the system is described but it also affects the performance of the co-simulation, sometimes even significantly. Hence, making this "knife-cut" demands careful consideration in relation to the required fidelity of the model.

11.5 Conclusion

We have briefly presented three industrial case studies that have used our approach. The ChessWay case has been discussed in detail in earlier chapters. For the new studies presented here—Verhaert's dredging excavator and Neoposts's document handling system—the use of co-modelling had positive technical consequences when compared to alternative approaches. These results are, of course, encouraging, but we also know from experience that the successful deployment of modelling technology requires more than a technical capability [84]. In Chap. 12, we record the lessons learned by Chess, Verhaert and Neopost about the impact on industrial development practice of introducing co-modelling and co-simulation using the Crescendo technology.