Collaborative Modelling and Co-simulation in Engineering and Computing Curricula

Peter Gorm Larsen[1], Hugo Daniel Macedo[1], Claudio Goncalves Gomes[1], Lukas Esterle[1], Casper Thule[1], John Fitzgerald[2], and Kenneth Pierce[2]

[1] DIGIT, Department of Engineering, Aarhus University, Aarhus, Denmark,
pgl@eng.au.dk hdm@eng.au.dk, claudio.gomes@eng.au.dk,
lukas.esterle@eng.au.dk
[2] School of Computing, Newcastle University, United Kingdom,
John.Fitzgerald@ncl.ac.uk Kenneth.Pierce@ncl.ac.uk

Abstract. The successful development of Cyber-Physical Systems (CPSs) requires collaborative working across diverse engineering disciplines, notations and tools. However, classical computing curricula rarely provide opportunities for students to look beyond the confines of one set of methods. In this paper, we report approaches to raising students' awareness of the integrative role of digital technology in future systems development. Building on research in open but integrated tool chains for CPS engineering, we consider how this has been realised in two degree programmes in Denmark and the UK, and give preliminary findings. These include the need for ensuring stability of research-quality tools, and observations on how this material is presented in Computing versus Engineering curricula.

1 Introduction

Collaboration between diverse disciplines is essential to the successful development of the Cyber-Physical Systems (CPSs) that are key to future innovations [26]. However, many university curricula train professionals within long-established disciplinary silos such as mechanical, electrical, civil or software engineering, with few opportunities for interaction between them. It is therefore critical to include elements within degree programmes to prepare students for the cross-disciplinary work that will likely feature in their subsequent careers [18]. The ideal is sometimes described as a *T-shaped* professional, with deep skills in one discipline, but an awareness of and capacity to work successfully with others.

The methods and tools to facilitate cross-disciplinary collaboration in CPS design are novel, and still the product of research. Engineering disciplines have developed vocabularies, ontologies, methods, notations and tools that are sometimes very different from one another, and are certainly different from those deployed in the newer disciplines of computing and software engineering. Promoting the kinds of collaboration needed to deliver modern CPSs is therefore challenging. In our work, rather than advocate a single formalism to be used across the diversity of forms of engineering in a project, we have seen some success from enabling stakeholders to use their own preferred notations and tools, but combining these in a semantically sound fashion. Our

universities have been involved in joint European H2020 projects (including DESTECS [5,10] and INTO-CPS [8,17]) for over a decade, leading to the development of tools that support collaborative model-based systems engineering of CPSs, with a particular focus on co-simulation [13]. The prime goal in this work has been to enable each collaborating stakeholder to continue to use their preferred technology, coupling the different models together using co-simulation.

Given the need for T-shaped professionals in CPS engineering, but also given the relative immaturity of methods and tools to address this challenge, the question arises: how can we use research products to inform the development of cross-disciplinary skills in university curricula? Although there is a widely-felt need to deliver research-inspired teaching in universities, it faces the challenge that proof-of-concept research products such as tools are often insufficiently stable to be used by newcomers. The risk is that students have a suboptimal experience when exposed to immature prototypes, potentially colouring their future attitude to novel and advanced techniques. In this paper we describe how such new research prototypes have been used in university education by ensuring that the prototypes is sufficiently stable to be used by novices.

Boehm and Mobasser [4] identify significant differences between the 'world views' of engineers specialising in physical systems, software and human factors, ranging from the approaches to economies of scale to forms of testing, as well as the underlying technical formalisms. They describe a curriculum and courses at Masters level that aim to broaden students' skills beyond software engineering alone to embrace T-shaped characteristics. This includes opportunities for students to undertake shared activities such as: developing shared operations concepts, jointly negotiating priorities and revisions with clients, jointly setting criteria for development approaches, determining risks, and many others. They identify tools to support systems thinking in these activities. In our work, we have focussed on newly emerging tools to support cross-disciplinary model-based engineering, and specifically, we aim to give students experience of T-shaped skills including: negotiating common terms and concepts across discipline models, identifying and performing system-level tests, modifying and reassessing designs, and performing design optimisation.

In this paper we examine two approaches to the incorporation of co-simulation into university curricula as a means of introducing students to the need for cross-disciplinary design. The first, applied at Aarhus University (AU) is to approach this through overall systems engineering at Masters level. The second, applied at Newcastle University (NU), addresses this in the context of a computing (mono-disciplinary) course at Bachelors level. We give an overview of the background in Model-Based Systems Engineering (MBSE) and our open tool chain that supports the collaborative approach outlined above (Section 2). We then describe how collaborative modelling and co-simulation has been introduced in the engineering curriculum at AU (Section 3 and the undergraduate computing curriculum at NU (Section 4). We discuss the experience so far in Section 5, and consider future directions in Section 6.

2 Background: the INTO-CPS Tool Chain

As Systems Engineering moves from document-based approaches to MBSE [30], the need arises to be able to analyse system models composed of diverse discipline-specific models, often from separate suppliers with their own intellectual property requirements. Co-modelling and co-simulation are seen as ways of meeting technical aspects of this challenge. The INTO-CPS tool chain shows how this can be realised.

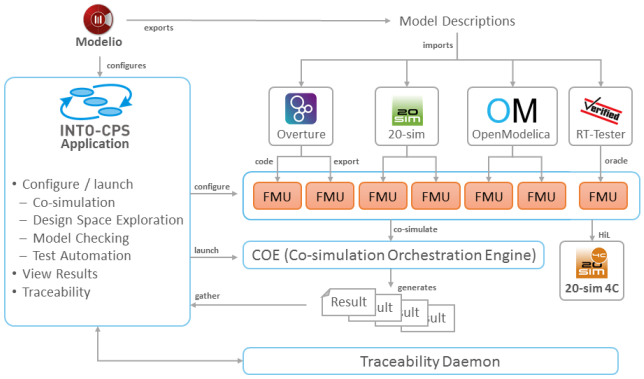

Fig. 1: Schematic of the INTO-CPS tool chain.

The INTO-CPS Tool Chain (Figure 1) supports an MBSE approach to CPS development that allows the Discrete-Event (DE) formalisms used to express cyber processes and Continuous-Time (CT) formalisms used for physical processes to co-exist in a common simulation framework. The act of collaboratively simulating these constituent models such that a simulation of a CPS is achieved is referred to as coupled simulation (co-simulation) [13]. A co-simulation is generally carried out by simulating the individual models while exchanging data and managing the progress of time between them.

Co-simulation requires the orchestration of a range of discipline-specific simulation tools [3,9]. The Functional Mock-up Interface (FMI) [22] is an approach to generalising the simulation interface of models that are to participate in a co-simulation. FMI provides and describes a C-interface and the structure of a model description file. A model implementing this C-interface and providing a model description file is referred to as a Functional Mock-up Unit (FMU) and it can contain its own solver. The INTO-CPS Toolchain is based on FMI 2.0 for its simulation capabilities.

The main user interface to the INTO-CPS tool chain is the INTO-CPS Application [20] which has been developed on the Electron platform.

The INTO-CPS Co-simulation Orchestration Engine (COE), called Maestro [27], manages the FMUs in accordance with various co-simulation algorithms. It is configured by a *multi-model* and a *configuration*. The multi-model defines the FMUs participating in a co-simulation, the dependencies between them, and their parameters. The configuration specifies the co-simulation execution including details such as logging and step size (i.e., the time interval between value exchanges between the FMUs).

The tool chain also supports Design Space Exploration (DSE). This is the process of systematically executing co-simulations with a variety of values for specified design parameters, with the goal of maximising an objective such as energy efficiency or a performance measure [6,9] and part of the INTO-CPS Toolchain.

The tool chain is neutral about the sources of FMUs, but it has been instantiated with several formalisms and simulation engines. For example, DE models can be developed in the Overture tool [16] which supports a dialect of the Vienna Development Method's modelling language for Real Time systems (VDM-RT [29]). Using Overture and the extension overture-fmu[3] one can export FMUs from a VDM-RT project [28,15]. CT models have been developed using 20-sim[4], which also can export conformant FMUs.

3 The Aarhus University Experience: Co-modelling and Co-simulation in the Systems Engineering Curriculum

In this section, we describe the introduction of co-modelling and co-simulation in the AU engineering curriculum. Students taking a Masters degree in electronic engineering or computer engineering follow a mandatory course in systems engineering. Formerly largely focussed on document-based processes, the course is evolving to provide a stronger introduction to model-based techniques, with about half of the course using the new research results in MBSE, multi-modelling and co-simulation through the INTO-CPS tool chain. Since the students are novices in the technology, this demands user friendliness in the prototype tools resulting from the research.

3.1 Course Structure and Content

The systems engineering course includes practical examples. A student driven project spanning the entire semester is performed in close collaboration with two Danish companies, namely Terma and Beumer. Here, student teams develop different systems engineering artefacts while the companies act as the respective customers in particular to embrace the T-shaped characteristics mentioned in Section 1 above. During the course the students learn on one hand to use different tools to model and design their systems and on the other hand to combine these tools in a single tool chain for engineering their systems efficiently using common terms. Hands-on tutorials as well as their ongoing group project guides them towards a deep understanding of MBSE and co-simulation.

To achieve this, the course is structured as follows: First the students get an overview on CPSs as well as co-simulation and the entire INTO-CPS tool chain. This overview allows us to structure the remainder of the lectures to dive into the relevant details. We use SysML, the industry standard for systems engineering, as graphical modelling language and VDM to formally specify software systems using the tools Modelio and Overture, respectively.

To teach students how to model dynamic systems, we utilise 20-sim, allowing students to define models via sets of equations or as graphical models, i.e., block diagrams.

[3] `https://github.com/overturetool/overture-fmu`
[4] `https://www.20sim.com`

Once students have the basis for constructing models of both cyber and physical systems elements, and have gained experience using industry tools such as 20-sim, they learn how to explore design parameters using the DSE features of the tool chain. This is followed by an introduction to the development of the software components, including automatic code generation.

3.2 Course Material

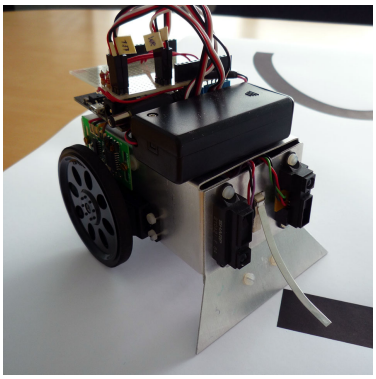

Fig. 2: Line Following Robot LFR used as a simple running example for co-simulation.

The INTO-CPS Association has created and maintains a series of tutorials on the installation and operation of the features of the INTO-CPS tool chain, using a simple running example based on the Line Following Robot (LFR) illustrated in Figure 2. These are:

Tutorial 0: Setting up the environment. This covers the installation of INTO-CPS, its main dependencies, the Java runtime, and the use of the COE Maestro.

Tutorial 1: First Co-simulation. This provides a demonstration of a water-tank co-simulation followed by an introduction to the concepts involved e.g. FMUs, models, multi-models, and an introduction on how to run a preset co-simulation.

Tutorial 2: Adding FMUs. This covers the deeper inner workings of a multi-model and co-simulation configuration, including the definition of parameters, which visual graphs to plot, which FMUs to use, and establishing the input/output connections between FMUs. Exercises help students to think about both the dynamics of the models being simulated and the effects of co-simulation parameters in the short- and long-term results of the co-simulation.

Tutorial 3: Using SysML. This introduces elements of SysML and the Modelio tool to develop the overall architecture of a multi-model. It covers the generation of a multi-model and its import into INTO-CPS.

Tutorial 4: FMU Export (Overture). The preceding tutorials have used a ready-made controller FMU. This tutorial explains where that FMU comes from. It includes the definition and export of a controller in VDM. Exercises allow the students to experiment with different control approaches.

Tutorial 5: FMU Export (20-sim). This focuses on the use of 20-sim to create and export the physical system FMU, allowing students to experiment with different physical models and observe how the controller behaves with each.

Tutorials 6 & 7: DSE configuration and execution. Building on Tutorial 3, students create a SysML representation of the DSE configuration (Tutorial 6) and learn how to use the DSE script to launch it and automatically compute fitness metrics for each design (Tutorial 7).

Tutorial 8: SysML for Co-Simulation. Students learn how use SysML to generate and import the description of the sub-components used in a co-simulation (Tutorial 3 was limited to using SysML to describe the overall multi-model architecture).

Tutorial 9: Building Controllers in VDM. This tutorial introduces techniques for designing controllers that are robust with respect to noise in the signals. Controllers are validated by co-simulation and are ready to be deployed.

Tutorial 10: Deploying the LFR Controller. At this point it is assumed that students have experienced how to develop the models of the LFR in a co-simulation (virtual) environment, and this illustrates how to deploy and validate a co-simulation unit using hardware performing in a real environment. The goal is to show how to export a VDM controller as an FMU, and subsequently use it to upload it as a sketch into the Arduino board controlling a hardware model of LFR.

Tutorial 11: Building Controllers in PVSio-Web. This tutorial teaches the students to write controllers in an PVSio-Web [21] and follows the approach in [23]. This tools enables the rapid prototyping of user interfaces and control code. The students then learn how to deploy their code into the hardware.

3.3 Course Delivery

We have applied the tutorials in the AU systems engineering course over the last three years. We group the tutorials into thematic units and deliver them throughout approximately five sessions depending on the students' progress. Sessions start with an exposition of the concepts behind the unit (e.g. co-simulation, DSE). Then hands-on tutorials allow students to interact with the tools while attempting to replicate the tutorial steps. A team of two to four teaching assistants helps troubleshoot problems students may find. Both software errors and repeated failed attempts to achieve the results of the steps prompt guidance using a case by case approach. Naturally, having newcomers using the prototype tools from the INTO-CPS project means that it gets used in ways not originally envisaged and this feeds back into further improvements of the tool chain elements. In the following, we provide an account of our experience of the sessions typical outcomes.

First session. This is the students' first contact with the model-based approach and co-simulation concept. Usually students fully complete Tutorial 0, 1 and 2 and achieve an abstract understanding a co-simulation. The first challenge appears with Tutorial 0 and the installation of the Java virtual machine, which surprisingly poses troubles to a great group of students. Although theoretically a simple step for a master level student to perform, in practice we observe that the students often get frustrated while finding the links for the software package in the official provider downloads webpage. The problem is

nowadays compounded with the display of complex licensing information and with the redirection to a registration page. It is awkward to observe that often students decide to install the virtual machine from non-official providers. This is a problem mostly affecting students without previous exposure to the Java and its development environment. In most cases, the completion of Tutorial 1 poses no trouble beyond tool glitches, or when the students miss accomplishing some of the previous steps. Commonly, a student may be unable to invoke the co-simulation orchestrator because they forgot to install the Java runtime or to download the `coe.jar` file. In contrast, the completion of Tutorial 2 is more complex and students often ask for assistance because the co-simulation is not launched (e.g., the misconfiguration of FMUs connections is detected by Maestro at launch time) or the results diverge from what is expected (e.g., the LFR animation displays a robot running in circles instead of following the line).

Second session. After a shallow contact with co-simulation, the students are presented with Tutorials 3 and 4. Tutorial 3 is usually appreciated as it provides an appealing graphical approach to manipulating and visualising the co-simulation multi-model. On the downside, at some points, the Modelio tool is not user-friendly. In some cases, it crashes unexpectedly and steps need to be repeated. This tutorial takes most students time, which causes some students to move the completion of Tutorial 4 to the following session. The completion of Tutorial 4 goes without much trouble. This may be because that most of the steps involve the use of INTO-CPS features that are already familiar from previous tutorials. We find that the novel aspect of it – the generation of an FMU using Overture – is typically less prone to tool crashes, and most of our students have previous experience with Overture, which may simplify the task.

Third session. This session is devoted to Tutorial 5 only. Installing and editing a CT model in 20-sim is usually accomplished with little assistance. In the second part, students are required to set up a C++ development environment in a Windows system, which involves the installation of tools such as Microsoft Build Tools and the troubleshooting of compilation errors. Usually students find this challenging and require more instructor support, for example because some students' systems do not accommodate the multi-Gigabyte demands of the installation, or the compilation script fails to find some of the registry entries.

Fourth session. At this point, students have a working knowledge of the DE and CT models of the LFR. The session consists of Tutorials 6 and 7 which explore the DSE concept. Some students follow Tutorial 6 with some difficulty, as the graphical language is too abstract. But most are able to finish and progress to Tutorial 7. This tutorial is usually well-received when completed, as the students have a hands-on experience with DSE. Students are usually able to configure DSE launch scripts, but often have trouble because of misconfiguration of the Python dependency and the specific libraries required to run the scripts.

Fifth session. To finalise the MBSE sessions we usually deliver one of the tutorials where the students develop the controller for the LFR example either using VDM (Tutorial 9) or PVSio-Web (Tutorial 11) and then deploy a controller into the hardware

platform Tutorial 10. Tutorial 10 involves the compilation and upload of an Arduino sketch into LFR. Some of the steps are cumbersome because the standard compiler in the Arduino IDE tool chain version must be substituted by a different version. Also, the process involves several options and flags, which may at points be confusing. The interested students get hold of the LFR hardware and deploy it and run it on line tracks we set up for the experiments.

To what extent have we addressed the need for the more T-shaped skills identified in Section 1? In the context of this systems engineering course, *negotiating common terms and concepts across discipline models* is exercised both in the document-based part as well as in the model-based part of the course. "Real" negotiation is actually carried out in the document-based part where the different groups also act as sub-contractors to another group. *Identifying and performing system-level tests* is in particular exercised in the model-based part where co-simulation is used in the tests performed. Experience at *modifying and reassessing designs* is done mostly in the document-based part where a design change is introduced (on purpose but) unexpectedly for the students. If we had more time we would like to also use this in the model-based part of the course. *Performing design optimisation* is exercised using DSE in a co-simulation context, and here we think that there is an opportunity to run a small competition to deliver designs that deliver optimal performance against specified systems-level criteria.

4 The Newcastle University Experience: Co-modelling and Co-simulation for Computer Scientists

NU's School of Computing admits about 300 students per year to study for a six-semester Bachelor of Science (Honours) degree in Computer Science, or an eight-semester Master of Computing degree. As a university focussed on fundamental research that has a positive business, societal or environmental impact, there is a strong motivation to expose students to advances in technology that are on the horizon now, but may become significant in their professional careers.

The focus of the NU computer science degree programmes has traditionally been on software and systems rather than on engineering. Perhaps as a consequence of this, although the programmes require a high level of attainment by students on entry, they do not require pre-entry qualifications to be in particular subjects. Students enter having specialised at high school in almost any discipline, although all must have a basic level of mathematics and around half do have backgrounds in mathematical and physical sciences. The resulting diversity of intellectual background among students is seen as a strongly positive feature, but it does mean that mathematical and computing maturity varies across the cohort on entry. Some "levelling up" in mathematics for computing takes place in the first semester of study, and the mathematics needed for specialised subjects, such as basic number theory for cryptography, is taught close to the point of use.

The undergraduate programmes considered here are taught over six semesters. We here consider the current Stage 3 specialist module in real time and CPS delivered in Semester 6. Students with an interest in this area will have studied some basic formal

modelling in VDM in Semesters 3 or 4, and may also have chosen to undertake a specialist project in the area in Semesters 5 and 6 alongside this pivotal module.

4.1 Course Structure and Material

CPSs typically have a significant real-time element as well as requiring the integration of physical and digital worlds. The MBSE approach taught at NU emphasises the real-time, concurrency and scheduling elements that Computing graduates need to know, alongside an appreciation of the multi-disciplinary cyber-physical integrations that give rise to temporal requirements.

The technical aims of the module are to understand the basic concepts of real time and embedded systems as well as CPSs; to understand the requirements and challenges of such systems, and how these have influenced the design of real-time languages; understand the implementation and analysis techniques for realisation of these systems; and to understand the concepts of model-based design, DE and CT models.

Introduction: The concepts of real-time, embedded and CPSs are clarified, and the integral role of dependability in such systems is introduced. In one classroom exercise, for example, students are asked to form teams responsible for the software, hardware or safety of a simple product (a personal transport device like a Segway). Concepts of larger-scale CPSs as systems-of-systems are introduced.

CT Modelling and Control of Physical Components: The motivations for CT modelling are introduced: this is particularly important for computer science students who have studied almost everything up to this point in an exclusively DE setting. In many cases, this is the point of reacquainting computer science students with physics and applied mathematics that they have not studied since high school. Concepts of controller characteristics (e.g., managing jerky acceleration) are illustrated using 20-sim and the standard example of the controller of a torsion bar in which a flexible axle connects two disks, one of which is rotated by a controlled motor. Elements of computer control (sample, compute and hold) are also introduced.

Discrete-event Modelling of Controllers: The idea of levels of control from loop to supervisory control are introduced by considering motorway driving. A review of VDM-RT includes a discussion of support for concurrency. Design patterns are introduced used as a basis for describing controller structures.

Multi-modelling and Co-simulation: The idea of a simple multi-model is introduced using the same LFR example as the AU course. This section discusses the pragmatics of co-model development and FMU integration. This is the point at which variable time step co-simulation semantics is first introduced. The full INTO-CPS tool chain is introduced, and DSE is first encountered, again using the LFR.

Dependability and Fault Tolerance: Key dependability concepts are introduced using traffic light control and a paper pinch control as examples. Students examine techniques for error detection, isolation and recovery and again use patterns to examine relevant solutions such as safety kernels and voter architectures.

Related Topics covered in the later classes of the module build on the core of control explored through co-modelling and co-simulation. For example, techniques for managing concurrency are discussed through the lock- and synchronisation-free

communications mechanisms that are needed for example soft control systems. Resource sharing is explored through the Mars Pathfinder priority inversion problem.

4.2 Course Delivery

The module is delivered over a single semester. It is expected to require around 100 hours of work from a student, of which about 36 are formal lectures or laboratory teaching. The remainder is independent study with access to labs, tools and learning materials including recordings of lectures. Contact hours are structured as a two-hour lecture followed by a one-hour practical session each week. This timetabling can be tiring, but it does offer the opportunity to try techniques discussed in the classroom immediately in the laboratory. Small items of practical work are embedded within the main course material, to familiarise students with specific technologies. These are followed by more substantial assessed coursework which builds on these smaller exercises and broadens students' experience. The module is assessed by a combination of this assessed work and a written examination.

Practical sessions begin once CT modelling and 20-sim have been introduced in lectures. Since the cohort are largely familiar with VDM and DE modelling[5] from Stage 2, this material is presented upfront because it is the most different to what students have previously experienced.

Practical classes are supported by three demonstrators (PhD students) and the lecturer. Each of the first four practicals introduces a tutorial exercise that lets students try out the tools and techniques discussed in preceding classes, reinforcing the content and providing students with the experience needed to tackle the longer assessed coursework. Each exercise is designed to take about one hour and must be signed off for a small amount of credit. Since students work at different rates and not all students can attend all sessions, sign-off sheets help to track individual students' progress; any students who seem to be falling behind are contacted individually to identify problems. Once all sign-off deadlines has passed, the final practical sessions allow students to ask for help on the longer piece of assessed coursework. In the following we discuss the practical exercises and student experiences. These have been refined over a few years and we highlight important changes that were made.

Installation. Practical classes are held in computer clusters with access to 20-sim, Overture and INTO-CPS, so students can start rapidly. One problem with INTO-CPS is due to the co-simulation engine using HTTP connections. Firewalls occasionally block this traffic, resulting in failure to run a co-simulation without a useful error message. One year there was a problem where only some machines had the correct firewall exceptions, so the problem was intermittent. Another issue was the need for the FMU export plug-in for Overture to be installed manually by each user, causing problems with roaming profiles. In one case, a new version of the plug-in was released that was incompatible with the version of Overture, so an entire practical was lost pending a workaround.

Students are increasingly using their own devices, and we provide installation instructions. Installation is typically smooth, but some machines have network-specific

[5] Exchange students unfamiliar with VDM are provided with extra gsupport.

firewall restrictions. In addition, some students using MacOS or Linux are unable to install the full tool suite (e.g., 20-sim). So, while students are not prevented from working, the nature of the tools might prevent some working in the way that suits them best.

Exercise 1 follows a lecture on CT modelling and simulation, and an introduction to 20-sim. Its aim is to familiarise students with the 20-sim interface (making connections, changing parameters and running simulations) by recreating a model of the standard torsion bar based on a screenshot of the layout. This exercise is based on training material from Controllab Products, with the of addition material to help clarify engineering concepts to non-engineer computer scientists. A problem here is that some students are not sure what they are modelling, since the torsion bar concept is not immediately familiar to most computing students. This might be improved by finding a more familiar example (e.g., a bouncing ball).

Exercise 2 introduces tuning of PID controllers, again using the torsion bar. This aims to help students understand the nature of low-level real-time control – one of the main new concepts the course introduces. The Ziegler-Nichols method, which is a common heuristic for tuning a certain class of system, is used to guide students. The torsion bar was designed as an example for teaching and is therefore fairly forgiving, which can cause confusion when students are unsure when they have finished tuning. In addition, while the Ziegler-Nichols method keeps the exercise straightforward, it does not provide good intuition of the roles of P, I, and D. This exercise could be enhanced using a different example where students tune manually before applying Ziegler-Nichols.

Exercise 3 covers DE modelling using Overture and the LFR example. As the majority of the cohort has studied VDM-SL, the aim here is to highlight the differences with VDM-RT. The exercise uses a "DE-first" version of the LFR example where a simple VDM environment model represents the robot. The produces output to the console and a CSV file that can be used to visualise the robot path in Excel. Students build a controller to follow the line, coping with ambient light. Some students find it frustrating building a controller without seeing the robot, however this is somewhat intentional as it motivates co-simulation in the next exercise. Another issue is that some students take time to understand the paradigm of control loops, for example using loops inside a main VDM operation, instead of treating the operation as the body of a loop. This exercise typically takes the most time.

Exercise 4 uses INTO-CPS to run a co-simulation using the LFR example. Students take their controller from Exercise 3 and see it driving the virtual 3D robot, which most students seem to find satisfying. This builds familiarity with the INTO-CPS interface, and how to run and monitor a co-simulation. The exercise is often quick to complete, compensating somewhat for the length of Exercise 3. A frustration here is that the older LFR model crashes when completing a co-simulation (even a successful one). Although a note is included in the exercise description, some students still find this confusing.

Assessed Coursework Rather than use smaller exercises, the assessed coursework focuses on a scenario that is explored in greater depth. For example, the 2019/20 course-

work is based on designing a controller for a driverless train and testing it using co-simulation. The train model includes a human passenger, the comfort of whose ride is important: the train cannot accelerate or brake too much if the passenger is to avoid falling. A model of train and passenger physics and a basic PID loop controller is given in 20-sim, and an outline supervisory controller in VDM-RT. A 3D visualisation is included to help test the controller. The tasks are:

1. Within 20-sim, tune the PID controller with respect to a permitted upper limit on passenger movement. Students explore a range of control parameters. Once tuned, the model is imported into the INTO-CPS application.
2. Create a VDM-RT model of a supervisory controller that manages train journeys subject to constraints on speed, stopping accuracy and passenger comfort. Students are given a base project in INTO-CPS that includes the necessary VDM-RT classes to read train position and speed, but which they may extend with features to structure the control logic as they see fit. An important requirement is the adaptability of the controller to a variety of train route scenarios.
3. Reflect on their experience of the task, considering how they went about tuning the controller, designing the controller to meet the constraints, and how their solution could be further improved.

4.3 Next Steps for the NU Curriculum

To what extent have we addressed the need for the more T-shaped skills that we identified in Section 1? Within the current NU module, students gain experience at *negotiating common terms and concepts across discipline models* from the outset because of the unfamiliarity of the kinds of CT model that we present. Concepts of controller architecture are, for example, quite new to students form this background. Experience at *identifying and performing system-level tests* is gained from the practical work. Experience at *modifying and reassessing designs*, and *performing design optimisation* is provided to some degree, e.g. in the coursework, but this is less systematic that we might wish.

The current module is moderately popular, attracting about 53 students in 2019-20 (about 20% of the available cohort). While those students who do take the course mostly report positive experiences, the relative unfamiliarity of the topic to classically trained computer scientists is potentially a deterrent.

In 2017 a complete review of the NU BSc computing curriculum was undertaken, influenced by two factors. First, there was a desire to expose students to active research topics earlier. Second, there was continuing recognition of the need to equip graduates with skills for employment [25]) in an increasing range of industries such as manufacturing that are not traditionally seen as destinations of software specialists [1]. The new curriculum takes a portfolio-based approach in which problem-based learning plays a significant role [2]. This suits CPS and MBSE perfectly. In order to ensure an introduction to research-inspired topics as early as practicable, a brief introduction to CPS will be given in Semester 4, giving students an opportunity to consider specialising in the area by taking the specialist module in Semester 5 and a capstone project in Semester 6. It is hoped that introducing topics in Stage 2 may demystify CPS engineering. The revised Semester 5 module will run in 2021-22 year with the following key changes:

1. The new Semester 4 module provides only a brief introduction to MBSE, CPS and VDM. This requires a change to the current delivery to teach more fundamentals at Semester 5, but it creates an opportunity to broaden students' MBSE experience in the CPS context by introducing a wider range of formalisms.
2. Two new academic staff will join the delivery team, bringing expertise in probabilistic modelling, machine learning and verification for CPS. This creates an opportunity to engage in further research-informed teaching and bring different perspectives on MBSE for a shared case study, for example.
3. The portfolio-based approach means students will be more familiar with larger pieces of coursework, team working and reflective writing. This creates an opportunity to expand assessment to more collaborative aspects such as assigning roles to students to create, share FMUs that could be integrated in assigned teams, or as a supplier-customer relationship between students supported by peer assessment and feedback to shape students' collaboration skills.

5 Discussion

In Section 1 we set ourselves the challenge of developing 'T-shaped' graduates, but doing so in the context of research-inspired curricula. The approaches we have taken at our two institutions are different in that one (AU) is situated in the context of a systems engineering course at Masters level, while the other (NU) is within a mono-disciplinary computer science programme at Bachelors level. In both cases we feel that this initiative can be considered successful if the graduates have both experienced the need for inter-disciplinary collaboration, and understood the need to develop and adapt professional practice as new research results become available, making companies innovative and competitive.

Although AU and NU have up to now placed the core modules on MBSE for CPS at different stages of study, it is notable that both institutions are now acting to place the first introduction of these topics earlier: both of them in Semester 4 of undergraduate programmes [14].

The NU module is part of a computer science degree delivered in a Computing school, rather than an Engineering degree in an Engineering education as at AU. This has influenced the content in that we are introducing students who have been thinking in largely discrete formalisms to the fact that their software will have – for good or ill – profound physical effects. Conversely, engineering students would benefit from greater awareness of the software engineering principles that will be critical to the success of innovations in many sectors. To that end, NU has created a new Masters programme in Smart Systems Engineering, aimed at both engineers and computer scientists.

Practical work plays a key role in both the AU and NU approaches to developing T-shaped skills for model-based CPS engineering. One of the most important lessons we have learned so far has been the need to create good ecosystems in which students practice and develop their skills. In our experience there are two such ecosystems to consider: first an ecosystem of disciplines; second, a business ecosystem in which roles such as contractors, integrators and end users are available. We have experienced pragmatic challenges in setting the disciplinary ecosystem up because discipline silos are

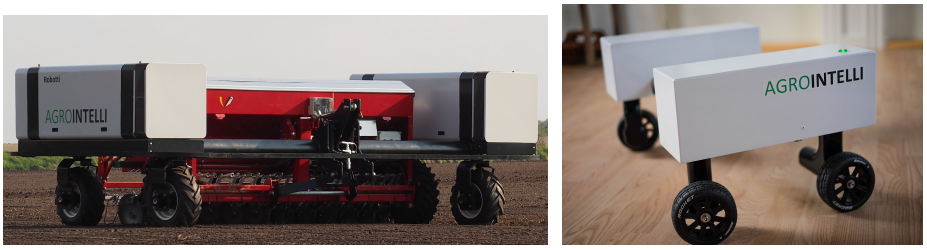

Fig. 3: The Robotti agricultural robot (left), and desktop-sized version (right).

often embedded in university structures. This makes it difficult to bring diverse groups together, often for prosaic reasons such as timetabling, but also because student and faculty expectations and forms of delivery differ between departments. As a result, we recommend developing an early-stage commitment to such multi-discipline projects. In creating a business ecosystem, we strongly recommend building relationships with external stakeholders to act as clients from outside the students' immediate technical environment. We have worked with real businesses, as well as other university professional departments for this purpose, chiming with the experience of Boehm and Mobasser [4]. The AU experience of creating subcontractors and integrator teams has been successful in creating a rich environment for understanding contractual relationships.

A lesson from both AU's and NU's experience is that successful deployment of research products in teaching hinges on having robust, well-documented tools with large bodies of examples aimed at users at a range of experience levels. This requires a very significant investment of effort in activities that rarely win academic plaudits. These include carefully structuring and refactoring tools, dealing with changes in platforms, developing, trialling, and improving materials (sometimes in several languages). Without these activities, tools and methods lack the credibility to influence more than a handful of the next generation of practitioners. Such an effort is typically only possible by maintaining a coherent series of research projects that keep key stakeholders involved.

6 Future Work

For 2021 we hope that we will be able to use a cloud-based version of the INTO-CPS Application [24] such that the students will have less installation necessary on their own laptops. In the future we hope to be able to use our research prototypes in a digital twin context [11]. Here we plan to make use of desktop version of the agricultural robot called Robotti [12]. This can be seen at Figure 3 and it is a platform that can be equipped with additional sensors and explored in a digital twin context as well [11].

In this paper, we have considered examples of the influence of research on teaching in MBSE for CPSs. The underlying idea is that the best way to have a positive influence on industry and the wider environment is to develop graduates who keep abreast of research and allow it to influence their professional practice. In that context, it is worth universities considering their responsibility for lifelong learning [7]. What can we do to maintain the skills and knowledge of own alumni, and maintain the virtuous cycle in

which graduates convey advances in practice to industry, which rewards universities in turn with new technological challenges for research and innovation.

At both AU and NU we have taken the initiative of establishing not-for-profit Digital Innovation Hubs (DIHs) with missions to improve the take-up of innovative technology in the surrounding business ecosystems, particularly in MBSE for CPSs. There is a long way to go in helping companies to truly take advantage of the expertise and innovations to be found in our universities. This is the goal of future work with partners in the HUBCAP project[6] [19] which aims to use DIHs to lower barriers to innovation through easier platform-based access to MBSE tools, models and practitioner experience.

Acknowledgements. We are grateful to many colleagues and students at both our universities. We acknowledge the European Union's support for the INTO-CPS and HUBCAP projects (Grant Agreements 644047 and 872698). We are especially grateful to the Poul Due Jensen Foundation, which has funded subsequent work taking co-modelling and co-simulation forward into the engineering of digital twins.

References

1. Made Smarter Review. UK Government. Department for Business, Energy and Industrial Strategy (2017)
2. Barnes, J., Colquhoun, J., Devlin, M., Heels, L., Lord, P., Marshall, L., Napier, C., Solaiman, E., Speirs, N., Talbot, L., et al.: Designing a portfolio-oriented curriculum using problem based learning. In: Proceedings of the 4th Conference on Computing Education Practice 2020. CEP 2020, Association for Computing Machinery, New York, NY, USA (2020), https://doi.org/10.1145/3372356.3372367
3. Bastian, J., Clauss, C., Wolf, S., Schneider, P.: Master for Co-Simulation Using FMI. In: 8th International Modelica Conference (2011)
4. Boehm, B., Mobasser, S.K.: System Thinking: Educating T-Shaped Software Engineers. In: Proc. IEEE/ACM 37th IEEE Intl. Conf. on Software Engineering. pp. 333–342 (2015)
5. Broenink, J.F., Larsen, P.G., Verhoef, M., Kleijn, C., Jovanovic, D., Pierce, K., Wouters, F.: Design Support and Tooling for Dependable Embedded Control Software. In: Proceedings of Serene 2010 International Workshop on Software Engineering for Resilient Systems. pp. 77–82. ACM (April 2010)
6. Broenink, J.F., Fitzgerald, J., Gamble, C., Ingram, C., Mader, A., Marincic, J., Ni, Y., Pierce, K., Zhang, X.: Methodological guidelines 3. Tech. rep., The DESTECS Project (INFSO-ICT-248134) (October 2012)
7. Field, J.: Social Capital and Lifelong Learning. The Policy Press (2005)
8. Fitzgerald, J., Gamble, C., Larsen, P.G., Pierce, K., Woodcock, J.: Cyber-Physical Systems design: Formal Foundations, Methods and Integrated Tool Chains. In: FormaliSE: FME Workshop on Formal Methods in Software Engineering. ICSE 2015, Florence, Italy (May 2015)
9. Fitzgerald, J., Gamble, C., Pierce, K.: Method Guidelines 3. Tech. rep., INTO-CPS Deliverable, D3.3a (December 2017)
10. Fitzgerald, J., Larsen, P.G., Verhoef, M. (eds.): Collaborative Design for Embedded Systems – Co-modelling and Co-simulation. Springer (2014), http://link.springer.com/book/10.1007/978-3-642-54118-6

[6] See hubcap.eu.

11. Fitzgerald, J.S., Larsen, P.G., Pierce, K.G.: Multi-modelling and co-simulation in the engineering of cyber-physical systems: Towards the digital twin. In: ter Beek, M.H., Fantechi, A., Semini, L. (eds.) From Software Engineering to Formal Methods and Tools, and Back - Essays Dedicated to Stefania Gnesi on the Occasion of Her 65th Birthday. Lecture Notes in Computer Science, vol. 11865, pp. 40–55. Springer (2019)
12. Foldager, F., Larsen, P.G., Green, O.: Development of a Driverless Lawn Mower using Co-Simulation. In: 1st Workshop on Formal Co-Simulation of Cyber-Physical Systems. Trento, Italy (September 2017)
13. Gomes, C., Thule, C., Broman, D., Larsen, P.G., Vangheluwe, H.: Co-simulation: a Survey. ACM Comput. Surv. 51(3), 49:1–49:33 (May 2018)
14. Hallerstede, S., Larsen, P.G., Boudjadar, J., Schultz, C.P.L., Esterle, L.: Frontiers in Software Engineering Education, chap. On the Design of a New Software Engineering Curriculum in Computer Engineering (2020)
15. Hasanagić, M., Fabbri, T., Larsen, P.G., Bandur, V., Tran-Jørgensen, P., Ouy, J.: Code generation for distributed embedded systems with vdm-rt. Design Automation for Embedded Systems (Nov 2019), https://doi.org/10.1007/s10617-019-09227-0
16. Larsen, P.G., Battle, N., Ferreira, M., Fitzgerald, J., Lausdahl, K., Verhoef, M.: The Overture Initiative – Integrating Tools for VDM. SIGSOFT Softw. Eng. Notes 35(1), 1–6 (January 2010), http://doi.acm.org/10.1145/1668862.1668864
17. Larsen, P.G., Fitzgerald, J., Woodcock, J., Fritzson, P., Brauer, J., Kleijn, C., Lecomte, T., Pfeil, M., Green, O., Basagiannis, S., Sadovykh, A.: Integrated Tool Chain for Model-based Design of Cyber-Physical Systems: The INTO-CPS Project. In: CPS Data Workshop. Vienna, Austria (April 2016)
18. Larsen, P.G., Kristiansen, E.L., Bennedsen, J., Bjerge, K.: Enhancing non-technical skills by a multidisciplinary engineering summer school. European Journal of Engineering Education (January 2017)
19. Larsen, P.G., Macedo, H.D., Fitzgerald, J., Pfeifer, H., Benedikt, M., Tonetta, S., Marguglio, A., Gusmeroli, S., Jr., G.S.: An Online MBSE Collaboration Platform. SimulTech 2020 (July 2020)
20. Macedo, H.D., Sanjari, A., Villadsen, K., Thule, C., Larsen, P.G.: Introducing Angular Tests and Upgrades to the INTO-CPS Application. In: Submitted for publication (2020)
21. Masci, P., Oladimeji, P., Zhang, Y., Jones, P., Curzon, P., Thimbleby, H.: PVSio-web 2.0: Joining PVS to HCI. In: Computer Aided Verification: 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I. pp. 470–478 (2015)
22. Modelica Association: Functional Mock-up Interface for Model Exchange and Co-Simulation. https://www.fmi-standard.org/downloads (October 2019)
23. Palmieri, M., Macedo, H.D.: Automatic Generation of Functional Mock-up Units from Formal Specifications. In: 3rd Workshop on Formal Co-Simulation of Cyber-Physical Systems (To appear). Oslo, Norway (September 2019)
24. Rasmussen, M.B., Thule, C., Macedo, H.D., Larsen, P.G.: Migrating the INTO-CPS Application to the Cloud. In: Gamble, C., Couto, L.D. (eds.) Proc. 17th Overture Workshop. pp. 47–61. Newcastle University Technical Report CS-TR-1530 (October 2019)
25. Shadbolt, N.: Shadbolt Review of Computer Science Degree Accreditation and Graduate Employability. UK Government. Department for Business, Innovation and Skills, and Higher Education Funding Council for England (2016)
26. Thompson, H. (ed.): Cyber-Physical Systems: Uplifting Europe's Innovation Capacity. European Commission Unit A3 - DG CONNECT (December 2013)
27. Thule, C., Lausdahl, K., Gomes, C., Meisl, G., Larsen, P.G.: Maestro: The INTO-CPS co-simulation framework. Simulation Modelling Practice and Theory 92, 45 – 61 (2019), http://www.sciencedirect.com/science/article/pii/S1569190X1830193X

28. Thule, C., Lausdahl, K., Larsen, P.G.: Overture FMU: Export VDM-RT Models as Tool-Wrapper FMUs. In: Pierce, K., Verhoef, M. (eds.) The 16th Overture Workshop. pp. 23–38. Newcastle University, School of Computing, Oxford (July 2018), TR-1524
29. Verhoef, M., Larsen, P.G., Hooman, J.: Modeling and Validating Distributed Embedded Real-Time Systems with VDM++. In: Misra, J., Nipkow, T., Sekerinski, E. (eds.) FM 2006: Formal Methods. pp. 147–162. Lecture Notes in Computer Science 4085, Springer-Verlag (2006)
30. Walden, D.D., Roedler, G.J., Forsberg, K.J., Hamelin, R.D., Shortell, T.M. (eds.): Systems Engineering Handbook. A Guide for System Life Cycle Processes and Activities, Version 4.0. Wiley, 4 edn. (January 2015)