Teaching the Fundamentals of the Modelling of Cyber-Physical Systems

Yentl Van Tendeloo
University of Antwerp, Belgium
Yentl.VanTendeloo@uantwerpen.be

Hans Vangheluwe
University of Antwerp, Belgium
McGill University, Canada
Hans.Vangheluwe@uantwerpen.be

ABSTRACT
Current Cyber-Physical Systems are becoming too complex to model and simulate using the usual approaches. This complexity is not only due to a large number of components, but also by the increasing diversity of components and problem aspects. In this paper, we report on over a decade of experience in teaching the modelling and simulation of complex Cyber-Physical Systems, at both McGill University, and the University of Antwerp. We tackle complexity through the use of multiple formalisms, each specialized for a specific domain and problem. Modelling and simulation is used throughout the complete development process. Students are introduced to all fundamental problems encountered when modelling and simulating Cyber-Physical Systems. Students will be able to both create a meaningful model in the formalism, as well as create a minimal simulation kernel for it. Our approach results in a deep understanding of the formalism, particularly the advantages and disadvantages, without focussing on tool-specific issues. In the end, students are capable of choosing the most appropriate formalism for a problem, and making an informed decision on which tool to use. Due to the variety of formalisms, students can successfully apply the gained knowledge in a wide spectrum of domains.

Author Keywords
Simulation; Modelling; Cyber-Physical Systems; Teaching

ACM Classification Keywords
I.6.0 SIMULATION AND MODELING: General; K.3.2 COMPUTERS AND EDUCATION: Computer and Information Science Education

1. INTRODUCTION
Current Cyber-Physical Systems are becoming too complex to model and simulate using the usual approaches. This complexity is not only due to a large number of components (often tackled by hierarchical decomposition), but is also caused by the increasing diversity of components and problem aspects. In order to minimize the accidental complexity exposed to the user, each component is best expressed in the most appropriate formalism for that individual component, instead of having "a single formalism to rule them all". Additionally, different forms of analysis are possible on a single model, resulting again in the use of different formalisms.

When teaching students about this domain, we wish to expose them to a wide variety of formalisms in a single semester course. Existing textbooks on this matter, however, are either too superficial (*i.e.*, cover a wide range of formalisms, but not in-depth), or too specific (*i.e.*, cover only a single formalism, but all aspects of it). Neither do they offer a broad view on different simulation paradigms, or on simulation semantics. For example, Law and Kelton [9] focus on the construction, configuration, and evaluation of experiments, but only lightly mention the wide variety of simulation formalisms in existence. Zeigler [20] focuses on the theoretical foundations of modelling, and in particular on Classic DEVS, making it difficult to understand from a practical point of view. Cellier [1] presents a nice overview of modelling and simulation, though is limited to continuous-time models, and mainly applied to physics. Fishwick [5] offers a multi-formalism view on the domain, but doesn't go in-depth about specific formalisms. Lee and Seshia [10] primarily focus on the modelling and simulation of Cyber-Physical Systems, but is mainly targeted at engineers through the high correlation with embedded platforms. Nutaro [12] goes into depth on the implementation of simulation semantics, but does not go to sufficient depth about theory and applications. To give students a complete overview of the domain, we prefer a course that touches upon a select number of formalisms, explained to sufficient depth, which can act as representatives for their domain. Depending on the interests of the student, supplemental courses can afterwards be taken for a more in-depth view on any of these formalisms.

In this paper, we report on over a decade of experience in teaching the modelling and simulation of Cyber-Physical Systems[1], both at McGill University and the University of Antwerp. We use modelling and simulation throughout the complete process, and for all different components, thus students will encounter all problems of Cyber-Physical Systems. Modelling and simulation is used for each individual component, and throughout the complete development process. Students are therefore introduced to all fundamental problems seen in the modelling and simulation of Cyber-Physical Systems. Information is gathered from different textbooks and

[1]The term "Cyber-Physical Systems was rarely used over a decade ago. Our course title and content also evolves over the years to keep up with advancements in this domain, though the core of the course has remained unaltered.

SpringSim-TMS/DEVS 2016 April 3-6, Pasadena, CA, USA

scientific papers, such as those mentioned above, to provide a consistent story about how all different formalisms relate. In the assignments, a single problem domain is explored with the use of all taught formalisms. All course material can be found online at **`http://msdl.cs.mcgill.ca/people/hv/teaching/MoSIS/`**.

Due to the broad application domain of our course, it became a mandatory course at the University of Antwerp for all computer science students. Its relevance for software engineers is obvious, but the course also serves as a fundamental introduction to modelling and simulation for students in data sciences, computational sciences, and computer networks. Follow-up courses are available for students, which relay the gained knowledge to the specific domain of the student.

2. COURSE DESIGN

Modelling and simulation is an integral part of the computer science curriculum at the University of Antwerp. Students will thus have already gained familiarity with some modelling languages, such as UML formalisms, Entity-Relation diagrams, Petri Nets, Finite State Automata, . . . which serve as a basis to our course.

Additional formalisms are taught, with simulation as the primary goal.

2.1 Prerequisites

To enroll in the course, students are expected to have some knowledge about the following concepts:

- **Object-oriented programming.** While the course focuses on modelling everything explicitly, basic programming knowledge is required for the students to understand the prototype simulation kernels, most of them developed in Python.

- **Patterns and anti-patterns.** Patterns simplify modelling by providing a kind of template to implement some frequently needed concept. A basic understanding of patterns and anti-patterns, as used in programming languages, is required to make a link to the modelling world.

- **Unified Modelling Language (UML).** Basic knowledge of the UML is necessary as a foundation for modelling, as well as for understanding the prototype tools. As the course focuses on simulation formalisms and the actual simulation, we expect students to already know about the basics of modelling as used in programming.

2.2 Objectives

Students should become familiar with different executable formalisms and their semantics (through simulation). We make a distinction between three kinds of formalisms, as shown in Figure 1, depending on how in-depth we go:

- **Simulation algorithm given with modelling assignment.** These formalisms are introduced to the full extent, and have additional information about how it is actually simulated using pseudo-code. While students are not asked to write their own simulator for these formalisms from scratch, they gain knowledge about the details of the formalism this way. Students gain insight into how different simulation paradigms can be simulated. For these formalisms, there are also assignments which require detailed knowledge of the semantics of the formalism.

- **Modelling assignment.** These are the remaining formalisms that are handled in detail, though their simulation algorithm is not presented extensively. We do not present these algorithms, as the mapping to code would be too time-consuming without any obvious advantage.

- **Briefly mentioned.** The final set of formalisms are only hinted at, or briefly mentioned throughout the theoretical classes. No assignments are prepared for them, as students are expected to be proficient with them through their previous experience with other formalisms.

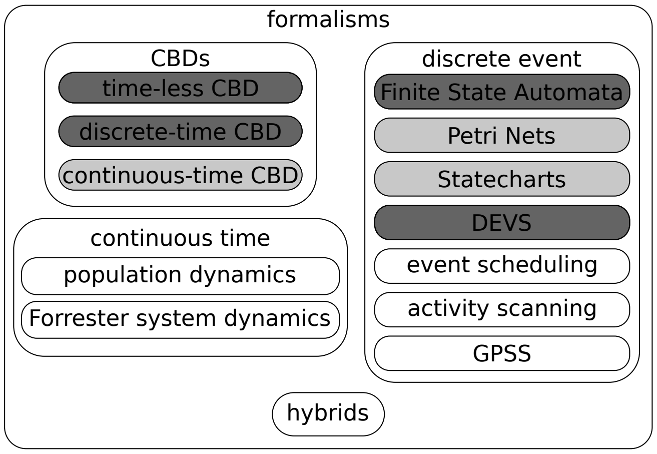


Figure 1. Formalisms taught in the course. White indicates formalisms which are only pointed to or mentioned, medium grey are formalisms with modelling assignments, and dark grey formalisms have a modelling assignment, as well as in-depth discussion about the simulation algorithm.

For each formalism, students should understand advantages, disadvantages, simulation semantics, as well as some basic modelling patterns for this formalism. We wish to present many formalisms within the allotted time, so lightweight prototype tools are used. These tools generally have a small learning curve, in contrast to the tools used in industry. Gained knowledge can be easily transposed to more advanced tools, as the knowledge will primarily be about the used formalisms, instead of the tool. They should be able to make an informed decision on which formalism, and which tool, to use for the modelling, analysis, simulation, and code synthesis of components of a cyber-physical system. Pointers are given to more formalisms (*e.g.*, activity scanning, Forrester system dynamics), extensions of taught formalisms (*e.g.*, coloured petri nets, parallel DEVS) and their combination (*i.e.*, hybrid formalisms).

2.3 Structure

The course has a weight of six European Credits (ECTS [3]), which are spread out over six core formalisms. One ECTS credit amounts to between 25 to 30 hours of workload for the students. In total, this means that students will spend up to 180 hours for this course, divided over theory lectures (36 hours), studying theory individually (54 hours), and making assignments in groups of two (90 hours each). Each formalism gets about the same amount of time: four to six hours of theory lectures, followed by an assignment in which the students learn how to use the formalism effectively.

The first few lectures serve as an introduction, where we explain why we need modelling and simulation for Cyber-Physical Systems, and what the specific problems are. The main focus lies on the increase of complexity in the development of Cyber-Physical systems. Students are introduced to the main causes of complexity: the size of the problem, interaction between different components, heterogeneity, emergent behaviour, and uncertainty. Modelling and simulation is offered as the solution to these problems.

After this general introduction to modelling and simulation, students are introduced to the different formalisms discussed before. For each formalism, a few theory sessions introduce the students to a brief history of the formalism, the problems it tried to tackle, and how it actually works. Formalisms are presented from two different views: the modelling point of view, and the simulation point of view. On the one hand, students need to be familiar with how the formalism is used in practice, as well as some common design patterns. This aspect is highlighted in the assignments, where students have to model a minimal system that still contains all essentials of the formalism. On the other hand, students should know how the model is simulated, making its semantics very clear. Apart from clearly defining the semantics, the complete simulation algorithm is presented for some formalisms. Students can then reason about the different corner cases and simulation performance. Students discover the weaknesses of the formalism, and are able to make their own basic simulation kernel.

In a final lecture, we take a step back and look at how different formalisms can be combined.

2.4 Evaluation structure

Course evaluation mainly consists of permanent evaluation of the assignments. Assignments are submitted through our university's electronic learning platform [16]. Late submissions are allowed, though points are subtracted. It is mandatory that students submit all assignments, even after the deadline has passed, to ensure that students have knowledge of each and every formalism.

Each assignment is made in groups of two, where students are expected to model a minimal system that touches upon the main aspects of the formalism. Grades are given based upon their knowledge of the formalism, and whether they use the most appropriate constructs and patterns. Afterwards, students individually defend their model and simulation results, where they are asked about some of their design decisions, as well as some possible alternatives to their solution.

The theoretical aspects are evaluated through a written theoretical exam at the end of the semester. Students are expected to present their knowledge on the theoretical definitions of the formalisms discussed (*e.g.*, formally write down the formalism), as well as the smaller semantic details of the formalism (*e.g.*, what happens in some corner case).

3. FORMALISMS

In this course, students are introduced to a wide spectrum of modelling formalisms. For several modelling paradigms, we picked out a select number of representative formalisms. Figure 1 presents these formalisms and the level of detail they are handled with.

The domain of the assignments change from year to year, while the core problems to be solved remain the same. To present a consistent story throughout the paper, we use this year's domain: railway simulation [14]. Each formalism tackles a different problem in this domain, showing that even in a single problem domain, multiple formalisms are beneficial. Figure 2 shows how they relate together. The remainder of this section discusses each assignment in detail: we briefly describe the problem statement, what kind of solution we expect, and the expected learning outcomes.

3.1 Introduction to multiple formalisms and modelling

In the first assignment, students start from a UML Class diagram [13], a set of requirements, and a long textual trace of the execution of the program. The program that generates the trace, however, contains a bug that the students should find through progressively moving from the Class Diagram and requirements, towards a testing framework. To do this, students map the UML Class Diagram to a UML Sequence Diagram, which represents the expected order of invocations. From this, a regular expression is generated that matches the behaviour indicated in the sequence diagram, by trying to match it with the textual trace. Finally, this regular expression is mapped to a Finite State Automata, for which we provide a very basic simulator. Through the simulation of this Finite State Automata, the bug becomes obvious as the Finite State Automata accepts the trace if it complies to the requirements, and rejects it otherwise. Figure 3 shows an overview of how all different formalisms are combined, to eventually arrive at a single yes or no answer.

The goal of this assignment is to show students that they are already familiar with several modelling and simulation formalisms, as none of these formalisms should be new to them. By going through all of these formalisms, students will also notice that there is no "best" formalism to tackle the problem: a combination of all formalisms should be used to arrive at the final solution. For students unfamiliar to Python, it provides a first example of how it will be used in the remainder of the course.

3.2 Discrete-time Causal Block Diagrams

In the second assignment, students expand upon a prototype Causal Block Diagram (CBD) [1] simulator by implementing some of the required simulation algorithms. This is a necessary prerequisite to implement the next assignment, where the formalism will actually be used. No modelling is done in

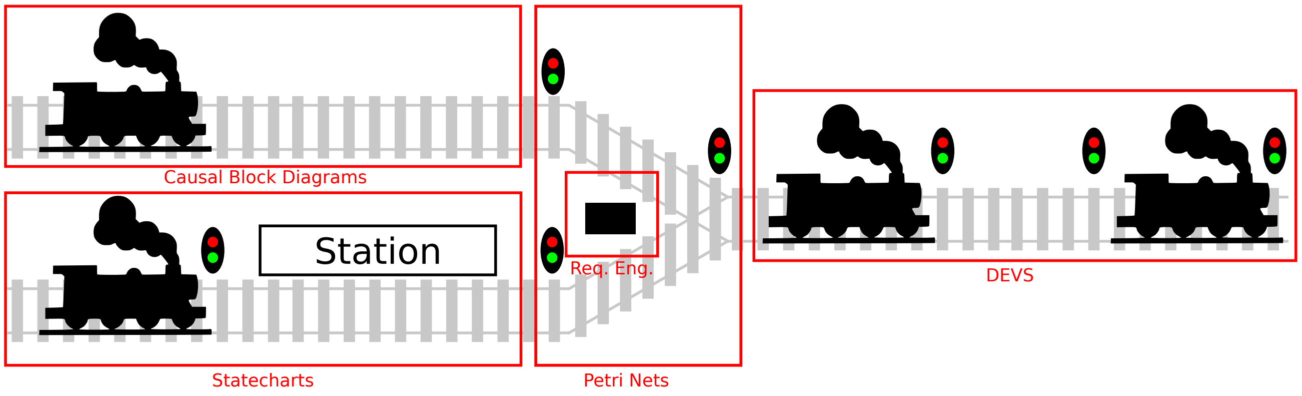
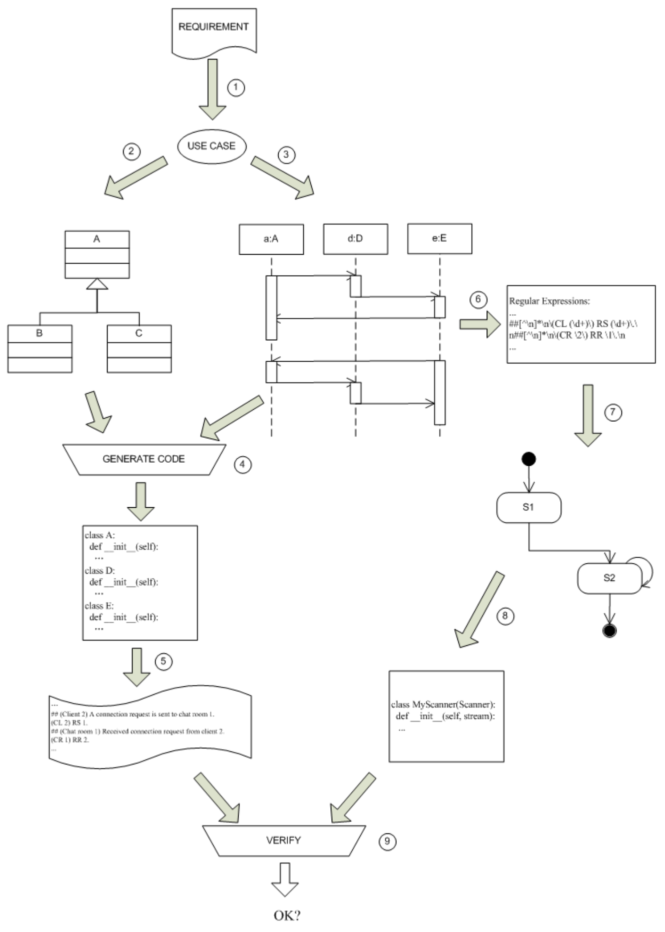


Figure 2. All formalisms used in the assignment are focussed around a single problem domain: railway simulation.



```
...
##[^\n]*\n\(CL (\d+)\) RS (\d+)\.\
n##[^\n]*\n\(CR \2\) RR \1\.\n
...
```

```
class A:
  def __init__(self):
    ...
class D:
  def __init__(self):
    ...
class E:
  def __init__(self):
    ...
```

```
...
## (Client 2) A connection request is sent to chat room 1.
(CL 2) RS 1.
## (Chat room 1) Received connection request from client 2.
(CR 1) RR 2.
...
```

```
class MyScanner(Scanner):
  def __init__(self, stream):
    ...
```

Figure 3. Overview of the first introductory assignment.

this phase of the assignment, though students analyze and extend an existing prototype simulator. The algorithms they implement are general graph algorithms, such as finding strong components in a graph. This allows students to easily grasp the algorithm being used, as they are already familiar with it.

The goal of this assignment is to teach how to analyze an existing simulation kernel, and complete its implementation. As the simulation kernel is minimal, students also find concepts that are common to many simulation kernels, such as an initialization phase, a while loop, tracing, and basic statistics gathering. Students can see how small a working (albeit naive) simulation kernel can be, making the task of creating your own simulation kernel seem less daunting. It also lets them combine time-less and discrete-time Causal Block Diagrams in their simulation kernel.

3.3 Continuous-time Causal Block Diagrams

Continuing on the previous assignment, students bridge from continuous-time Causal Block Diagrams to discrete-time Causal Block Diagrams. This is achieved by mapping the two new blocks — integrator and derivator — over to the discrete-time domain through discretization. Additionally, they model a small Causal Block Diagram for implementing the controller of an autonomous train. The controller causes the train to accelerate and decelerate, trying to match the ideal velocity defined for the specified segment. The displacement of the passengers should be plotted over time, to show how it evolves in the case of different ideal velocities and different strategies in the controller. All formulas are given, and need to be mapped over to Causal Block Diagrams. Simulation results of a specific scenario are given in Figure 4, showing how the displacement evolves in function of the time.

The goal of this assignment is to show how different formalisms are related, and that continuous systems can also be modelled through discretization. By manually applying the discretization, students become aware of what happens internally in the simulation tool, as well as understanding the risks of discretization, both in their own tool, as well as in other tools. Thanks to the implementation of the small model, they gain valuable knowledge on the modelling of continuous systems, as opposed to discrete event systems in the remainder

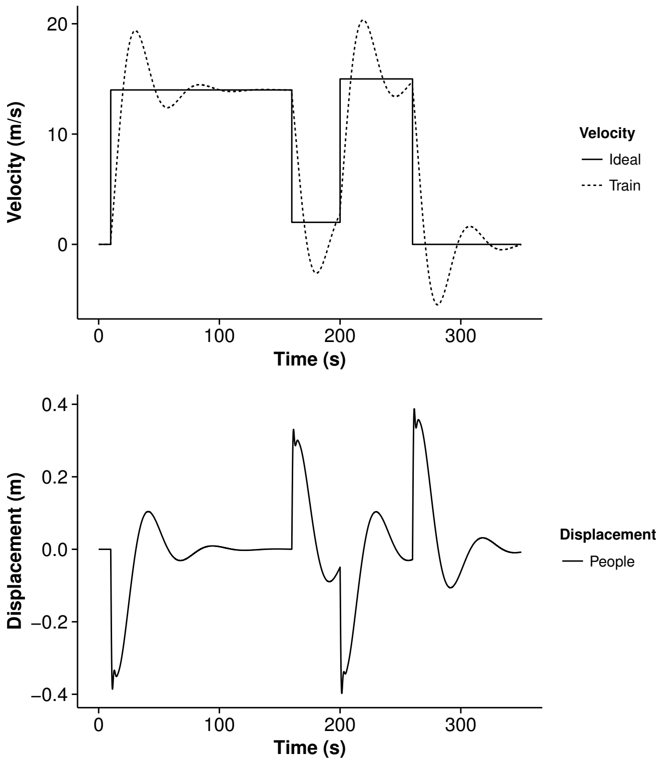


Figure 4. CBD simulation results.

of the course. As students have to present their simulation results, they also learn the basics of tracing a simulation run.

3.4 Petri Nets

For the Petri Nets assignment, students model a basic safety-critical system in Petri Nets [11], using the PIPE [2] tool. Students have to guarantee the safety of a simplified railway junction in the presence of two trains. To allow for all interleavings, students should manually guarantee fairness of the system, while keeping some non-determinism around. Basic analysis on the model needs to be done, such as showing liveness, safety, find invariants, generate the coverability graph, . . . These results should be related back to the problem domain. For example, if the Petri Net deadlocks, then what are the implications on the system under study?

The goal of this assignment is to show the power of Petri Nets for analyzing non-deterministic systems. By manually guaranteeing fairness, and using weighted transitions, anti-places, and inhibitor arcs, students show that they are able to apply basic patterns in Petri Nets. Students notice that the formalism is rather verbose: even for the minimal model they need to implement, the size causes it to become unreadable very fast. Analyzing the Petri Net provides insight in the capabilities of Petri Nets, but also its limitations. By going to the coverability graph, they notice that Petri Nets are even able to handle unbounded states. The problem of state space explosion is also briefly exposed to the students. Furthermore, it highlights the difference between simulation and verification, as the used Petri Net tool supports both. Students are therefore asked to do both simulation and analysis of the Petri Net.

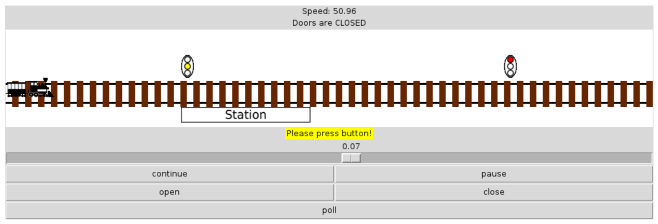


Figure 5. Example User Interface that is controlled by a statechart, which the students have to write.

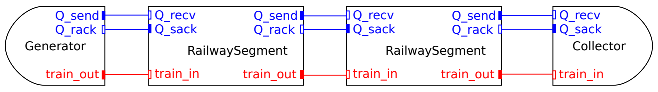


Figure 6. The DEVS assignment as a queueing problem.

3.5 Statecharts

In the Statecharts assignment, students will model the timed, reactive behaviour of a simple user interface, using AToMPM [15], backed by a Statecharts [6] compiler [4]. The user interface, shown in Figure 5, is a simplified train control panel, which controls the acceleration, the doors, and a dead man's switch that needs to be pressed every so often. Beyond that, there is also a pause and continue button, which controls the simulation itself. While the user interface is conceptually simple, as it consists of only five buttons and a slider, implementing its behaviour in code is non-trivial due to the interplay of concurrent and timed behaviour. Low-level operations, such as computing the new velocity, are provided to the students. Finally, the statechart is used to synthesize code out of the model, which can be executed stand-alone.

In the theory lectures, students are introduced to the semantic variation points between tools like STATEMATE [8] and Rhapsody [7]. This knowledge needs to be related back to the assignment.

The goal of this assignment is to bring the use of modelling and simulation closer to the domain the students are familiar with: writing code. Students notice that their models are not only useful for obtaining information about the system under study, but that it can also be used to automatically generate code that implements the modelled behaviour. This bridges the final gap between modelling and simulation, and execution. The Statechart model contains all special constructs that students should be familiar with: composite states, orthogonal states, history states, and basic event passing.

3.6 DEVS

In the final assignment, students model and simulate a Classic DEVS [20] model, using the PythonPDEVS [19] tool. Students model performance of different railway configurations: all of them are just straight tracks, but there are a variable number of traffic lights in between segments, influencing the length of a single segment, and thus the size of the "critical section". With some abstractions, this model can be reduced to the one shown in Figure 6, which resembles a queueing model. Using this performance model, students can determine the average transit time for different configurations. As a traffic light locks the complete segment after it, more traffic

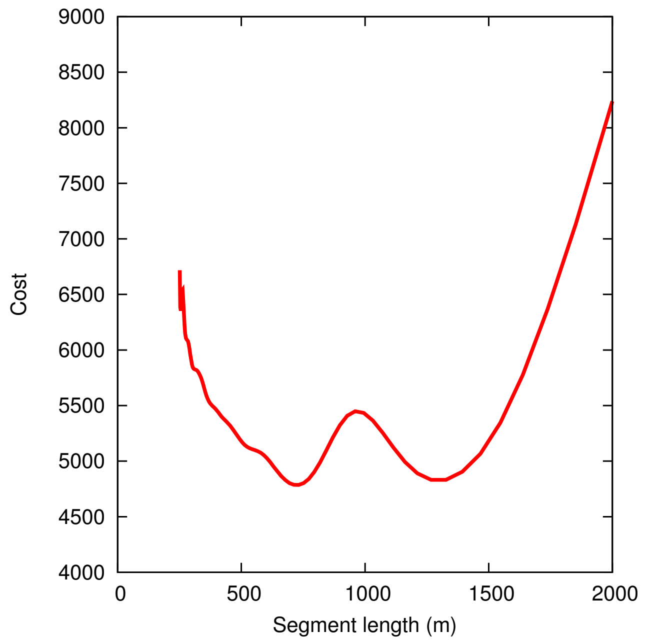


Figure 7. Example results for the DEVS assignment.

lights allow for more trains on the same straight track, while still remaining safe. However, traffic lights require maintenance and an initial investment. Students are given a cost function which defines the trade-off between average transit time, and maintenance costs. Using the cost function, students plot the cost of all different configurations and need to say which choice is optimal. Students write their own experiment, which performs statistics gathering and tries to find an optimal solution. Example simulation results for a specific configuration of train characteristics is shown in Figure 7, where the ideal distance (*i.e.*, with the lowest total cost) between two traffic lights is around 700 meters.

The goal of this assignment is to make students familiar with performance modelling, and subsequent optimization of the model. Students understand that the proposed problem would be too complex to solve through other means, and too expensive to do trial-and-error experiments. Students become familiar with the use of cost functions, and the possibility to do optimizations using them. The DEVS model contains the following special constructs that students should be familiar with: the different kinds of transitions (internal vs. external), different models (atomic vs. coupled), accessing global simulation time, basic event passing, and use of the select function. Some experience is gained in writing experiments, introducing them to the world of design space exploration.

4. LESSONS LEARNED

Throughout more than a decade of teaching this course, we learned many lessons on how to efficiently teach this topic. We summarize our findings in the form of best practices. Throughout the years, the contents of the course have slightly moved to better adapt to the current needs of industry, as well as to stay up to date with current academic advancements in modelling and simulation. We also present some future improvements that we would like to change, or at least inspect, in the future. Finally, the industrial relevance of our course is assessed.

4.1 Best Practices

We agree to, and apply, the best practices identified by [17, 18]. Some of them, however, have been altered due to the large amount of formalisms that students are exposed to:

- **Expert supervision.** Due to the many distinct formalisms, it is best to have several teaching assistants, each specialized in the formalism being used. In our case, the teaching assistant for each topic has contributed to the tools being used, or has used it to a greater degree than required by the students. It is this teaching assistant that creates the assignment and assesses feasibility too, due to the knowledge of common pitfalls in the formalism, as well as how the formalism is used in practice. The introduction to the tool is given by this teaching assistant, due to the intricate knowledge of the features used during the assignment.

  However, there are only very short practice sessions. In these sessions, the tool is briefly introduced and an example is shown. After this, students are free to experiment with the tools themselves. This approach is possible because the tools are stable and known to work well. The features of the tool are rather restricted, making controls obvious. Another advantage is that this motivates students to actively look at the source code of the simulation kernel, and understand what is going on. Of course, the teaching assistant is still available for questions and problems.

- **Start small.** We agree that it is best to start with a small example, which students would be able to finish in the course of a single practice session, followed by a bigger assignment students should work on afterwards. However, this is not feasible due to the large amount of formalisms students are exposed to. Instead, students are only expected to make a simple assignment, which is in itself small. While students are not exposed to the problems caused by large-scale models, it still offers them sufficient background to understand the formalism, and be able to tackle bigger problems if need be. They do become familiar with the main problems of modelling in the formalism though. So in contrast to [18], we limit ourself to small assignments, in order to cover more formalisms. In the context of our course, we favor a **minimal assignments** best practice (discussed below) over this one.

We identified some new best practices too, motivated by our experiences:

- **Provide further reading.** While the theory lectures provide an introduction to the formalism and its goals, students are offered a set of papers that introduce the formalism. Apart from these mandatory papers, some optional papers are presented as well, which go deeper about details, applications, or possible extensions of the formalism. These optional papers contain valuable additional information about the formalism, which helps students to enhance their knowledge about the formalism in question. To reduce the workload of students, however, these papers are not part of the theoretical exam, and we only expect students to use them as background material.

- **Minimal assignments.** Again due to the high number of formalisms covered, the assignments should be as minimal as possible. Minimal, however, does not mean superficial: the assignment should still contain problems that require an intricate knowledge of the formalism to solve. The knowledge gained from solving the minimal assignment should also be transferable to bigger projects. While the modelling of realistic systems provides benefits too, such as being closer to real applications of the formalism, as well as finding out how well the formalism scales, this is not an option for our purposes due to the limited time available for each formalism.

- **Tool agnostic.** The assignments should focus on the formalism, and not on tool-specific features. This allows students to appreciate the subtleties of the formalism, instead of the tool's implementation details. As our teaching style is tool agnostic, students are able to model systems independent of the tool used. Knowledge from the prototype tools is portable to all similar tools, as it only focuses on the formalism itself, which is similar in all tools.

- **Provide simulation algorithm.** By providing the simulation algorithm, students gain intricate knowledge about the semantics of the formalism. Since they know the details of the algorithm, they understand corner cases, and what the fundamental limitations of the formalism are. Furthermore, they understand how simulation kernels are implemented, giving them an advantage should they need to write their own simulation kernel in the future.

- **Minimal open-source tools.** By using minimal tools, students are not distracted by the plethora of features offered by tools used in industry. While these features are useful in big projects, they only act as a distraction for our small assignment, increasing accidental complexity. As tools are open-source, students can inspect the source code to find the simulation algorithm, that was presented in class. To extend the understanding of the formalism, we have students implement parts of the simulation algorithm.

- **Follow-up Master's thesis topics.** As we touch upon so many formalisms and application areas, students are sure to find a topic relevant to their domain of interest. Offering a wide range of Master's thesis topics in all of these directions attracts these students to increase their knowledge of the domain. Thanks to these topics, some of our minimal prototype tools have grown to full-blown tools that have contributed to the scientific community.

- **Follow-up courses.** Since modelling and simulation is an integral part of the curriculum at the University of Antwerp, several follow-up courses are offered. One such course is about Model Driven Engineering, following the format of [17, 18]. In this follow-up course, the level of abstraction is raised even further by going to domain-specific formalisms. The formalisms taught in our course are then used as the semantic domain of these domain-specific modelling languages.

4.2 Future Improvements

Even though this course has been given for more than a decade, it still continuously changes to adapt to the needs in industry and academia. We summarize some of the improvements we plan to make to the current form of the course:

- **Couple all assignments together.** Currently all assignments focus around the problem domain of railway simulation. Each assignment, however, models a different part of the domain. So while it is conceptually clear that a single domain needs all these different formalisms, it would be beneficial to couple all assignments together into a single application. Apart from showing the students that all methods are orthogonal, instead of alternatives, it also introduces them further to the problem of composition of modelling formalisms.

- **Prepare for follow-up courses.** At the end of the course, it might be advisable to explain how the concept is continued throughout the remainder of the curriculum. In particular, the different follow-up courses could briefly be introduced, with an explanation as to how they relate to our course. Students would have clearer knowledge of how this course fits in the curriculum. Also, motivated students can choose to take follow-up courses that they deem interesting with their gained knowledge about modelling and simulation.

- **Relate to tools used in industry.** It could be beneficial for students to have a brief *industry* lecture, where their current knowledge is related to industry tools. This could show students that their knowledge is easily transferred to industrial tools, as well as show them the additional features that these tools offer. While a single lecture might not be sufficient to go in-depth in all tools, it will motivate students to get to know these tools better.

4.3 Industrial Relevance

From an industrial point of view, our approach allows students to clearly understand the core of modelling and simulation. As there is no focus on any tool in particular, students will be able to quickly grasp the basics of most similar tools used in industry. Students also have a deeper understanding of the particularities of specific tools, and are therefore in a better position to give companies technical advice in which formalisms and tools to use.

As they gained experience in writing parts of a simulator themselves, students will be capable to extend existing industrial simulation tools. This skill is particularly useful for simulation tool builders. Consequently, many of our students end up at such companies.

5. CONCLUSION

We presented our current way of teaching a course on the modelling and simulation of Cyber-Physical Systems, guided by over a decade of experience. Our course combines the theoretical foundation of modelling with a wide spectrum of modelling and simulation formalisms. We touch upon UML, as well as the Causal Block Diagrams, Petri Nets, Statecharts, and DEVS formalisms. The theoretical underpinning is given such that students become familiar with a family of formalisms instead of a single one. Sufficient details are given about the formalism, such that students will be able to implement their own simulator, but also are able to effectively use it for modelling. In the assignments, students employ this knowledge to model a simple component, which touches upon common problems in that formalism. Where applicable, parts of the simulation tool are left open for students to fill in the gaps. This teaches students to think from both points of view: as a modeller, as well as a simulation tool builder.

ACKNOWLEDGMENTS

This work was partly funded by a PhD fellowship from the Research Foundation - Flanders (FWO). Partial support by the Flanders Make strategic research centre for the manufacturing industry is also gratefully acknowledged. We would like to thank the other teaching assistants of this course: Bart Meyers, Cláudio Gomes, and Joachim Denil.

REFERENCES

1. Cellier, F. E. *Continuous System Modeling*, first ed. Springer-Verlag, 1991.
2. Dingle, N. J., Knottenbelt, W. J., and Suto, T. PIPE2: A tool for the performance evaluation of generalised stochastic petri nets. *SIGMETRICS Performance Evaluation Review 36*, 4 (Mar. 2009), 34–39.
3. European Commission. ECTS – European Credit Transfer and Accumulation System. `http://ec.europa.eu/education/ects/ects_en.htm`, 2015.
4. Exelmans, J. Statecharts and class diagrams. `http://msdl.cs.mcgill.ca/people/joeri`, 2015.
5. Fishwick, P. A. *Simulation Model Design and Execution: Building Digital Worlds*, first ed. Prentice Hall PTR, 1995.
6. Harel, D. On visual formalisms. *Communications of the ACM 31*, 5 (May 1988), 514–530.
7. Harel, D., and Kugler, H. The rhapsody semantics of statecharts (or, on the executable core of the uml). In *Integration of Software Specification Techniques for Applications in Engineering*, vol. 3147 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2004, 325–354.
8. Harel, D., and Naamad, A. The STATEMATE semantics of statecharts. *ACM Transactions on Software Engineering Methodology 5*, 4 (Oct. 1996), 293–333.
9. Law, A. M., and Kelton, D. M. *Simulation Modeling and Analysis*, third ed. McGraw-Hill Higher Education, 1999.
10. Lee, E. A., and Seshia, S. A. *Introduction to Embedded Systems, A Cyber-Physical Systems Approach*, second ed. 2015. `http://leeseshia.org`.
11. Murata, T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE 77*, 4 (1989), 541–580.
12. Nutaro, J. J. *Building software for simulation: theory and algorithms, with applications in C++*. John Wiley & Sons, 2011.
13. Object Management Group. Unified Modelling Language (UML). `http://www.omg.org/spec/UML/2.5/`, 2015.
14. Pachl, J. *Railway operation and control*, third ed. VTD Rail Publishing, 2014.
15. Syriani, E., Vangheluwe, H., Mannadiar, R., Hansen, C., Van Mierlo, S., and Ergin, H. AToMPM: A web-based modeling environment. In *Proceedings of MODELS'13 Demonstration Session* (2013), 21–25.
16. University of Antwerp. Blackboard academic suite. `https://blackboard.uantwerpen.be/`, 2015.
17. Van Gorp, P., Schippers, H., Demeyer, S., and Janssens, D. Students can get excited about formal methods: a model-driven course on petri-nets, metamodels and graph grammars. In *MoDELS Educators' Symposium* (2007), 1–10.
18. Van Gorp, P., Schippers, H., Demeyer, S., and Janssens, D. Transformation techniques can make students excited about formal methods. *Information & Software Technology 50*, 12 (2008), 1295–1304.
19. Van Tendeloo, Y., and Vangheluwe, H. The modular architecture of the Python(P)DEVS simulation kernel. In *Proceedings of the 2014 Symposium on Theory of Modeling and Simulation - DEVS* (2014), 387–392.
20. Zeigler, B. P., Praehofer, H., and Kim, T. G. *Theory of Modeling and Simulation*, second ed. Academic Press, 2000.