Getting Software Engineering out of Isolation

(Invited Paper)

Peter Gorm Larsen and Joey W. Coleman
Aarhus University
Department of Engineering
Finlandsgade 22, DK-8200 Aarhus N, Denmark
e-mail: {pgl,jwc}@iha.dk

John Fitzgerald
Newcastle University
School of Computing Science
Newcastle upon Tyne, NE1 7RU, UK
e-mail: John.Fitzgerald@ncl.ac.uk

***Abstract*—We argue that the main challenges to be overcome in developing future generations of IT-enabled products and services lie not so much in the software engineering discipline itself as in the collaborative relationships that software engineers have with other disciplines. We briefly review the need for more emphasis on multi-disciplinary approaches and consider three classes of demanding system: embedded products, systems-of-systems and cyber-physical systems. In each of these areas, we argue that there is a need for engineering with formal semantic bases that enable joint modelling, analysis and simulation of groups of heterogeneous models.**

***Index Terms*—Formal methods, multi-disciplinarity, modelling, simulation, systems-of-systems, cyber-physical systems, embedded systems**

I. Introduction

The discipline of "software engineering", the origins of which are traceable to the NATO conferences of the late 1960s, emerged from a perceived need for a well-founded discipline underpinning a technical response to a software crisis [1], [2]. Unfortunately, the success rate of software development projects is still far from ideal [3], [4]. We would speculate that the rise of software engineering as a separate discipline has brought a real risk that software developers are distanced from the other engineering disciplines involved in product development. It almost seems a truism to say that software engineers need to understand well the system in which their product will operate, but the separation of engineering teams and phases in development still militates against this. The growth in embedded systems, and in networked integrations of software-rich systems, makes it imperative for software engineers to find ways of developing systems collaboratively with disciplines that may use quite radically different modelling and analysis techniques. This implies global system models that encompass cyber and physical elements, enabling the analysis and trade-off of decisions across the boundaries between these domains.

The mono-disciplinary nature of most engineering educations contributes to the challenge here. Graduates trained as experts in one single discipline may have difficulties in understanding stakeholders from different disciplines and the challenges that they face. As a result, solutions that are globally optimal from a system-wide perspective are not reached because modelling and analysis work tends to focus on mono-disciplinary optimisation. There is a need in our education system to ensure that students get at least a feel for the significant design parameters from other disciplines.

One example of such an opportunity for collaborative engineering is between control and software engineering. Modelling embedded systems that are intended to affect the physical world is a significant challenge [5]. Models of the physical world often involve differential equations that describe how the world changes over time; these models are based on continuous mathematical domains. Models of digital controllers –often software-based– usually do not reference time at all and instead treat the model of the controller in terms of discrete events that trigger specific reactions from the controller; these models are based on discrete mathematical domains [6]. In order to be able to appropriately balance concerns from control engineering with software engineering and taking potential faults into account a common model that can be analysed is needed [7]. The challenge can be further extended when different systems can provide added benefits for its users by interaction with other systems. There is a new emerging discipline for the proper engineering of such System of Systems (SoSs). Here the challenges cannot be solved using software engineering or system engineering principles alone. The scaling here gives new research challenges that clearly can get inspiration from existing engineering disciplines but a multidisciplinary approach is necessary again.

In this paper we look at the need for multidisciplinarity in engineering education in Section II. In Section III, we focus on how software engineering aspects might be combined with consideration of the physical world surrounding software. Section IV discuss scaling software development up to the level of "Systems-of-Systems". Section V takes this further on to cyber-physical systems that have aspects of both embedded systems and systems-of-systems. Section VII provides concluding remarks and identifies future research directions aiming at getting software engineering out of isolation.

II. Multidisciplinary Engineering Education

Most university studies are highly specialized within a single discipline –to the point of being strictly mono-disciplinary at some universities– in order to impart a sufficient depth of skill to the students. This approach is true of most European universities and many North American institutions as well.

As a contrast, there are universities, mostly in North America though some are in Europe, that follow the liberal arts tradition. These institutions require that students take courses from outside of their chosen discipline. This applies mostly to programs offered at the Bachelour's level, as Master's level degrees remain specialized even at liberal arts institutions.

At all levels of education, however, it is important that the students have some significant exposure to challenges and terminology from other disciplines [8]. Without this exposure there is a substantial risk that students starting their careers will face challenges in understanding and collaborating with peers and stakeholders from different disciplinary backgrounds. The question is how to best prevent this situation from happening, and how to provide the appropriate tools for students to overcome this situation when it does occur. Practical experience suggests the direction of a broader answer to this question.

A summer school with the aim of providing students with a multi-disciplinary angle to their studies has run since 2008 in Denmark, in collaboration with Bang & Olufsen [9]. The summer school is entitled "Conceptual Design and Development of Innovative products". Students with different disciplinary and cultural backgrounds are combined in mixed groups. Each group then receives assignments that require them to use their combined skills to solve it appropriately. This requirement for collaboration is rooted in problem-based learning principles [10]. The results of delivering this summer school series has been so successful that the idea is being exported and expanded with a similar summer school in China in 2012.

A similar multidisciplinary summer school for PhD students was delivered for the first time in Portugal in 2012, titled "Innovation and Creativity for Complex Engineering Systems". Here the focus was more on producing multidisciplinary research plans. This summer school is also project-based but more teacher-led lecturing was included to give the students a better understanding of specific topics, such as how to write multidisciplinary research plans.

The challenge is to get such stand-alone events incorporated into the standard curriculum used in (engineering) educations such that all students get exposed to this kind of experience during their studies. We believe that a first start of this is to ensure that all (engineering) students get a joint course on system engineering [11]. However, we believe that collaboration across disciplines and respect for the challenges in other disciplines will only fully be achieved if the students try to solve multi-disciplinary assignments in multi-disciplinary groups.

In the spirit of the liberal arts traditions seen in Bachelor's education, we would ultimately like a similar "technical arts" tradition for specialist Master's degrees. This is most important for those programs whose topic inevitably leads to cross-disciplinary collaboration. This proposal is not to introduce a "general science year" into the curricula, but rather to ensure that the graduates of specialist Master's programs have significant exposure to the terminology and challenges of other scientific disciplines.

The incorporation of a stand-alone multidisciplinary project course –tacking problems that require input and skills from many disciplines– into the regular curriculum is a first step towards a technical arts tradition. The main challenges for implementing these courses are mainly of practical administration since such courses typically need to be coordinated between different disciplinary studies and across different departments. However, if it was successfully implemented the students would get out of isolation of their own discipline and as a consequence be much better at solving more complicated challenges at the general system level when they would enter their professional careers.

III. Embedded Systems

An embedded system is a computer system designed for specific control functions within a larger system, often with real-time computing constraints [12]. It is embedded as part of a complete device often including hardware and mechanical parts and typically with less interface towards human users. By contrast, a general-purpose computer, such as a Personal Computer (PC), is designed to be flexible and to meet a wide range of end-user needs. Embedded systems control many devices in common use today.

The DESTECS project[1] has taken a first step in the direction of crossing between different disciplines necessary in the embedded control domain [13]. This project addresses collaborative, multidisciplinary design of embedded systems using methodology and tools that promote rapid construction and evaluation of well-founded system models.

One of the main impediments to the design of embedded real-time control solutions is the separation of engineering disciplines. While control engineering typically uses tools operating on Continuous-Time (CT) models, software engineering is founded on Discrete-Event (DE) models. In order to evaluate alternative designs and support early defect analysis or correction, it is essential that engineers collaborate across disciplines in short windows of opportunity [14], [15]. Model-based approaches provide a way of encouraging collaboration, but engineers need to jointly perform design evaluation and analysis using models expressed in different tools. These tools should reflect the relevant aspects of the design in a natural way, but also allow consistent, rapid analysis and comparison of models. Achieving this requires advances in CT modelling; formal DE modelling of controllers and architectures; fault modelling and fault tolerance; and open tools frameworks. These various advances are the aim of the DESTECS project.

An Example: Train Carriage Braking

It is simply impossible to develop a useful controller without close interaction between the disciplines of control engineering, software engineering, mechanical engineering and electrical engineering. Each of these will have their own established modelling techniques and formalisms. As an example, consider the development of software for controlling the speed of railway carriages by applying the brakes. Associated with

[1] "Design Support and Tooling for Embedded Control Software" http://www.destecs.org

each carriage, control software takes account of environmental conditions (e.g. current speed, temperature, friction etc.) and fixed design parameters (e.g. number and position of wheels, mass of the carriage etc.) in order to determine how best to apply the brakes on command from the driver or safety unit. It only makes sense to talk about the behaviour of this software in the context of the product of which it is a part – the railway carriage as a whole.

Imagine one scenario for the development of the braking system. Well-established control laws for this type of braking system are developed using tools based on continuous time models (perhaps using numerical solutions to differential equations as a simulation). These laws are passed to software developers who discover that the laws cannot be directly implemented on the processors available because certain calculations necessary for processing the data from the sensors cannot be completed quickly enough within the processor schedule. By this stage in the development process it may be too late to modify the carriage design, or use alternative and better sensors. The CPUs for the controller software will often have been fixed and even purchased some time previously, so they cannot be replaced with faster processors. The only remaining option may be to modify the schedule running on the processor, so that some other functionality affecting less critical functions, like the smoothness of the ride, for example, have to be rescheduled, compromising performance and the quality of the product. In practice, there can often be a slow iterative process in which control laws are re-engineered and re-implemented several times before a compromise is reached, reducing time to market.

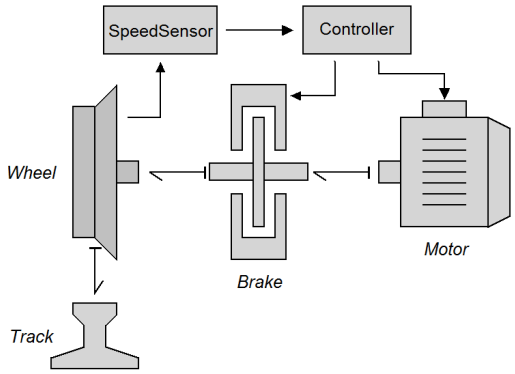

Fig. 1. A 20-sim model of a train carriage braking built from bond graphs and iconic diagrams.

If the developers could model the control laws collaboratively and *at the same time as* the software, some of these difficulties might be reduced. For example, an alternative choice of sensor might be made, replacing the original design with one that does pre-processing of the data, and this could be checked out and evaluated at the modelling stage. In DESTECS, we build such collaborative models (*co-models*) which represent the semantic integration of the software design, based on DE computation models in VDM, with the CT models of the plant and control laws in a bond graph formalism supported by the 20-sim environment (see Figure 1). For our example, such a co-model would contain:

- A CT model encompassing the wheels of the carriage, their friction and force linkages to the track, and the braking mechanism itself.
- A DE model of the control software, including the main control loop. This may also include the supervisory control software which manages system functionality a level above the loop control, including the switching of modes, error detection and recovery. In our experience, supervisory control accounts for 80% of the software content of an embedded product, and can be a greater source of potential defects than the loop controller itself. The DE model can readily be expressed in VDM with its real-time extensions. In our example, supervisory control might be responsible for switching alternative control laws into force if the temperature of the brakes exceeds a certain value, or invoke emergency braking modes if signalled to do so from the train driver's cab.
- An interface (in DESTECS this is termed the *contract*) between the two models defining the shared design parameters that both sides need to know, the shared variables that are monitored or controlled and any events of interest. Events are logical predicates that may cause the simulation of the CT-side model of the plant to be interrupted in order so that a response can be generated by the controller (for example if a sensed value crosses a threshold).

The question of where to place the models of sensors and actuators is interesting. In many cases it is appropriate to describe them "CT-side", but digital control might sometimes suggest placing them DE-side. The co-model as a whole presents an interface to a co-simulation engine that allows the CT and DE models to be executed together. The co-simulation engine implements a reconciled operational semantics of the two models, managing the synchronisation of time and state between them. The co-simulation can proceed under the control of a script that implements a particular scenario in terms of the actions of the environment (for example in raising a braking signal), and the invocation of fault models that may be built in to the DE or CT models. The exploration of the space of design alternatives is enabled by such co-simulation. Multiple scenario-based tests can be used to assess the performance of either alternative plant or controller designs. In our example, this could include an assessment of alternative numbers and configurations of sensors (modelled CT side) with a appropriate changes of control loop (DE-side). Being able to perform design space exploration at an early stage is the essence of successful system design, and the example emphasises the extent to which this is a multi-disciplinary activity enabled by software engineers work in collaboration with others, and certainly not alone. The challenges that remain in such cross-disciplinary assignments are much more complex and important to tackle than those that remains inside software engineering itself.

IV. Systems-of-Systems

Modern network technologies are enabling the integration of pre-existing computing systems into "Systems-of-Systems" (SoS) that together deliver a service that the constituent systems could not offer alone [16]. Examples include emergency response systems formed from the coalition of information systems of the response services such the ambulance, hospital and fire services. The public who expect responsive emergency services may demand confidence that the SoS will deliver safe, rapid and secure transfer of patient data between these systems. Examples on another scale might include the audio-video ecosystem in a home in which digital content is streamed from multiple sources to multiple users via a range of systems provided by different manufacturers. The customer experiencing the SoS expects to have a consistent experience (such as a common playlist) as they move through the ecosystem from device to device. The manufacturers of the devices also need to demonstrate that they will respect the digital rights agreements in force for the content as it is played through their devices, even though they may be delivered through another system in the SoS.

While embedded systems are often characterised by closed loop control, SoS are more general, are distributed, and typically have more human interaction. The distinguishing characteristics of SoSs are:

- **Operational independence:** A SoS is formed by heterogeneous constituent systems, many of which may not have been originally designed for participation in the SoS. They may be described using a wide range of methods and require modification, for example, through wrapping or linking interfaces, in order to achieve integration
- **Managerial independence:** The constituent systems may be managed independently and so can change functionality or character during the life of the SoS in ways that are not foreseen when they are originally composed.
- **Distribution:** Constituent systems may be distributed and decoupled, and yet a communications infrastructure should support the protocols necessary to facilitate coordination between them.
- **Evolutionary development:** The independence of constituent systems means that the SoS changes over time to respond to changing goals or component characteristics.
- **Emergence:** SoSs exhibit behaviour that their components do not exhibit on their own.

In a rather conventional view of software engineering, software systems are constructed in a highly directed way by teams who are usually within the same organisation, and who (at least on paper) have a shared understanding of the goals of the system being developed. The components can be designed to use carefully defined interfaces that violate their independence by revealing data and services in order to manage their collaboration. The operational and managerial independence characteristics mean this is not the case. One category of SoS ("directed" SoS [17]) does allow for a master that has the power to get owners of other constituent systems to adapt to their wishes, but SoS that adhere to this structure are comparatively rare. In general, the characteristics listed above pose major challenges to conventional "closed" software engineering methods if we wish to develop SoS that are dependable. Independence means that developers can have only limited knowledge of, and confidence in the likely behaviour of constituent systems. Distribution (in some cases also mobility) can compound the difficulty of gaining confidence in concurrent behaviours. The need to manage evolutionary development means that some ability to cope with change must be built-in. Above all the reliance on emergence means that the verification of global SoS-level behaviour must follow from the composition of the behaviours of individual constituents. As our emergency response and audio-video examples show, in practical terms, SoS Engineering is challenged by the large number and range of stakeholders (the owners and operators of the constituent systems as well as the users who experience the SoS as a whole). Here again, software cannot be developed in isolation and thus software engineers need to be able to think in a SoS setting where they cannot control all parts.

The COMPASS[2] project that aims to develop systematic engineering principles that are applicable to SoS design, including the scaling-up of modelling and validation techniques from a formal methods to address the SoS challenges. COMPASS is defining the first formally based modelling language specially designed to target the SoS area [18]. The challenge of constituent system independence is addressed by recording contracts that express our limited knowledge about constituents, constraining, but not completely prescribing, their range of behaviours. Global SoS-level properties are verified as the composition of the properties guaranteed in the constituent system contracts.

Example: Interoperable Train Carriage Systems

An important objective of development work in the rail sector is to increase the interoperability of railway equipment such as carriages from different suppliers, so that it becomes possible to mix them in the same "set" or train, for example by coupling trains from different railway systems at national borders, to form larger trains.

A train made up from a heterogeneous collection of component carriages is a form of a SoS, and Figure 2 gives a partial representation of this. The constituent systems are the information and computing systems in each carriage. These constituents were not necessarily designed with the intention of being in a mixed set, and they will generally be running software that is managed and upgraded by suppliers who are independent of each other. They are networked in the train and geographically distributed, and the software in each manufacturer's carriage can change with time. In spite of all this, we expect them to deliver a consistent emergent experience to the passengers on board. The developers of train systems are unlikely to subjugate their own corporate

[2]This is an acronym for "Comprehensive Modelling for Advanced Systems of Systems".

goals to those of the SoS, so this is not a "directed" SoS. We would not risk simply plugging carriages together, so some level of explicit cooperation is needed, even at the level of ensuring data transfer, so this is not a "virtual" SoS either [16]. Rather, depending on the degree of explicit cooperation, it is an "acknowledged" or a "collaborative" SoS.

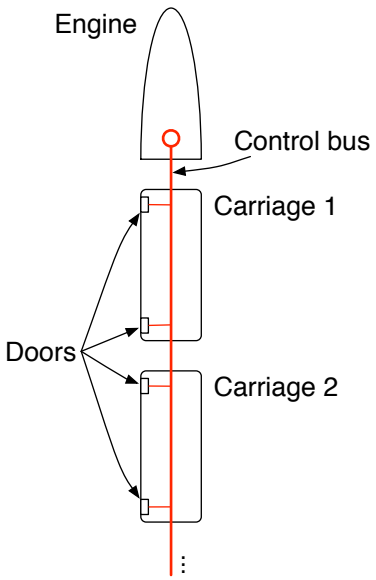

Fig. 2. Diagram of a system of train carriages.

A model of this SoS contains contractual descriptions for each of the services offered at the boundaries of each carriage and other constituent systems. Such contracts involve assumptions and guarantees. For example, the contract on a door locking service of a constituent system might record a guarantee to lock the doors within 3 seconds of receiving a specified signal; the assumption might be that the signal comes with a valid carriage identifier. Contracts also record the constraints on interaction behaviours, recording for example that a "Lock the doors" command is acknowledged once the doors have been locked, or a special response denoting failure if the locks did not work. Furthermore, the door locking services of all carriages must communicate with the control bus on each carriage, and each carriage's control bus must cooperate with the busses on connected carriages and with the control system in the engine.

In COMPASS, we work on formalisms for contracts that support the description of both functional and interaction behaviours. Verifying a global property, such as the fact that all of the train's doors will be locked, or failure acknowledged within a specified time, should follow from the contracts offered by the constituents. For such a SoS we require not merely that data should be syntactically compatible between units, but that there should be a semantic mapping, so that "Lock the doors!" is interpreted in similar ways in all carriages of the train. In SoS engineering, as in embedded systems, software engineers have to deal with heterogeneity of systems and models, and cannot confine their analysis to mono-disciplinary approaches. This again provides a need to think outside the isolated software engineering discipline.

V. Cyber-Physical Systems

Cyber-Physical Systems (CPS) are integrations of multiple computing and physical processes, with the potential to design and adapt both computing and physical elements to improve efficiency and resilience of the system as a whole [19], [20]. This encompasses the conventional control systems which typically are represented in a static setup. However CPS can also be seen in increasingly dynamic settings. Examples of mobile cyber-physical systems include applications to track and analyse Carbon Dioxide emissions [21], detect traffic accidents and provide situational awareness services to first responders [22], measure traffic [23], and monitor cardiac patients [24].

The engineering challenges associated with developing dependable CPS combine those of embedded systems and systems-of-systems. The ultimate goal here is to be able to analyse trade-offs of design alternatives that cross the boundaries between cyber and physical elements, as with embedded systems, but also between multiple cyber and multiple physical elements. As with SoS, the integrator has only limited knowledge and confidence about the constituent computing and physical elements, so these can only be modelled in contractual terms. In a CPS setting, we need to consider a range of interfaces between physical and cyber elements, including force or other physical phenomena, and not merely data. Indeed, the field of rigorous engineering methods for dependable CPS is still in its infancy [25].

Example: Controlling the Train!

In our multi-carriage train example, the SoS formed by distributed control of the diverse carriage braking systems forms a CPS. Developers requiring to verify that braking commands will result in the train speed reducing within a specified distance or time need to consider the multiple cyber control elements in the constituent carriages, and their different capabilities, as well as the effects of braking on the train physics. In this latter aspect, the physical elements have an influence on one another through the braking force of the train. Aside from verifying safety-related properties, having a CPS multi-model based on a number of networked co-models, allows us to analyse properties such as energy consumption in different operating scenarios. The kinds of design trade-off that might be considered are variations in the braking command parameters sent to different carriages to take account of their physical differences as well as the differences in their control characteristics. Aside from the multiple co-modelling involved, there is a significant amount of research to be done in visualisation of scenario outcomes, and the associated support for design space exploration, in such a complex system.

VI. Related Work

The ideas presented here build for example on advances in embedded systems design and fault modelling. In the embedded sector, BODERC [26] developed a method to predict

performance of real-time control systems, albeit with little tool support for trade-off studies or co-simulation. Modelica [27] is an object-oriented, equation-based multi-domain language for simulating controlled physical systems, and provides source libraries of physical components similar to the approach taken in the DESTECS project. Approaches to co-simulation of discrete-event and continuous-time models have been defined by Nicolescu et al. [28]. Ptolemy II [29] offers both discrete-event and continuous-time simulation within a single tool, though lacking the object-orientation offered by VDM and the component libraries offered by 20-sim. Work on time synchronisation between DE and CT models is described in *hybrid systems* literature, for instance, Cassandras et al. [30]. The DESTECS approach is distinctive in including a rich but abstract DE-side modelling language, and in managing co-simulation of heterogeneous models in their "native" tools.

Collaborative modelling is essential in both SoS and CPS engineering. Several approaches to collaborative modelling are described by Renger et al. [31]: problem structuring methods focus on the decision-making process including simulation for scenario exploration; group model building takes extends the conceptual model to simulation models to explore different options; enterprise analysis focuses on models that are built collaboratively. Our work is focussed on collaborative construction of models that are then used to explore design options.

VII. Concluding Remarks

Software engineering is a maturing discipline with sound scientific foundations, a range of methodologies, increasingly robust tools, and a wealth of experience. However, in this paper, we have argued that the complex products, solutions and services developed in the future, and the levels of dependability that they demand, require a more multidisciplinary and collaborative approach. This has consequences for software engineering technology, for formal methods themselves, and for training and education, all of which need to cross an increasingly broad range of disciplines and modelling types.

Our experience in the DESTECS and COMPASS projects has been a first small step in this direction. There remain opportunities for significant advances in modelling technology, including the areas of semantics, tool support, design space exploration, methodology and model management. The need to package relevant research results as industry-ready methods and (open) tools is paramount. We hope that the ideas presented in this paper will cause more researchers to carry out their research in software engineering in a larger context in order to have a more significant impact on the actual development of software-based systems in industry. Hopefully there will be fruitful collaborations for this kind of multidisciplinary research.

Acknowledgements

Both the DESTECS and the COMPASS projects have been supported by the European Commission under the 7th Framework programme. We would like to thank our collaborators from the DESTECS and COMPASS projects for their work to make the joint visions become reality. Finally we would like to thank Nick Battle, Carl Gamble and Claus Ballegaard Nielsen for providing valuable input on drafts of this paper.

References

[1] P. Naur and E. B. Randell, "Software Engineering: Report on a Conference sponsored by the NATO Science Committee," garmisch, Germany, 7th to 11th October 1968, Brussels, Scientific Affairs Division, NATO, January 1969.

[2] W. W. Gibbs, "Software's Chronic Crisis," *Scientific American*, pp. 72–81, September 1994.

[3] The-Standish-Group, "The Chaos Report," http://www.projectsmart.co.uk/docs/chaos-report.pdf, 1995.

[4] J. Johnson, *My Life Is Failure: 100 Things You Should Know to Be a Better Project Leader*. Standish Group International, 2004.

[5] T. Henzinger and J. Sifakis, "The Discipline of Embedded Systems Design," *IEEE Computer*, vol. 40, no. 10, pp. 32–40, October 2007.

[6] A. Tiwari, N. Shankar, and J. Rushby, "Invisible Formal Methods for Embedded Control Systems," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 29–39, January 2003.

[7] M. Verhoef, "Modeling and Validating Distributed Embedded Real-Time Control Systems," Ph.D. dissertation, Radboud University Nijmegen, 2009.

[8] S. Jahanian and J. M. Matthews, "Multidisciplinary Project: A Tool for Learning the Subject," *Journal of Engineering Education*, April 1999.

[9] P. G. Larsen, J. M. Fernandes, J. Habel, H. Lehrskov, R. J. Vos, O. Wallington, and J. Zidek, "A Multidisciplinary Engineering Summer School in an Industrial Setting," *European Journal of Engineering Education*, August 2009.

[10] D. L. Maskell and P. J. Grabau, "A Multidisciplinary Cooperative Problem-Based Learning Approach to Embedded Systems Design," *IEEE Transactions on Education*, vol. 41, no. 2, pp. 101–103, May 1998.

[11] R. Stevens, P. Brook, K. Jackson, and S. Arnold, *System Engineering – Coping with Complexity*. Pearson Education, 1998, vol. ISBN 0-13-095085-8.

[12] D. D. Gajski, S. Abdi, A. Gerstlauer, and G. Schirner, *Embedded System Design, Modelling Synthesis and Verification*. Springer, 2009.

[13] J. F. Broenink, P. G. Larsen, M. Verhoef, C. Kleijn, D. Jovanovic, K. Pierce, and W. F., "Design Support and Tooling for Dependable Embedded Control Software," in *Proceedings of Serene 2010 International Workshop on Software Engineering for Resilient Systems*. ACM, April 2010, pp. 77–82.

[14] J. Fitzgerald, P. G. Larsen, K. Pierce, M. Verhoef, and S. Wolff, "Collaborative Modelling and Co-simulation in the Development of Dependable Embedded Systems," in *IFM 2010, Integrated Formal Methods*, ser. Lecture Notes in Computer Science, D. Méry and S. Merz, Eds., vol. 6396. Springer-Verlag, October 2010, pp. 12–26.

[15] J. Fitzgerald, P. G. Larsen, K. Pierce, and M. Verhoef, "A Formal Approach to Collaborative Modelling and Co-simulation for Embedded Systems," *To appear in Mathematical Structures in Computer Science*, 2012.

[16] M. W. Maier, "Architecting Principles for Systems-of-Systems," *Systems Engineering*, vol. 1, no. 4, pp. 267–284, 1998.

[17] J. Dahmann and K. Baldwin, "Understanding the Current State of US Defense Systems of Systems and the Implications for Systems Engineering," in *IEEE Systems Conference*. IEEE, April 2008.

[18] J. Woodcock, A. Cavalcanti, J. Fitzgerald, P. Larsen, A. Miyazawa, and S. Perry, "Features of CML: a Formal Modelling Language for Systems of Systems," in *The 7th International Conference on System of System Engineering*, July 2012.

[19] J. White, S. Clarke, C. Groba, B. Dougherty, C. Thompson, and D. C. Schmidt, "R&D Challenges and Solutions for Mobile Cyber-Physical Applications and Supporting Internet Services," *J. Internet Services and Applications*, vol. 1, no. 1, pp. 45–56, 2010.

[20] E. Lee and S. Seshia, *Introduction to Embedded Systems, A Cyber-Physical Systems Approach*. University of Berkley: http://LeeSeshia.org, 2011, iSBN 978-0-557-70857-4.

[21] J. Froehlich, T. Dillahunt, P. Klasnja, J. Mankoff, S. Consolvo, B. Harrison, and J. A. Landay, "UbiGreen: Investigating a Mobile Tool for Tracking and Supporting Green Transportation Habits," in *Proceedings of the 27th international conference on Human factors in computing systems*, ser. CHI '09. New York, NY, USA: ACM, 2009, pp. 1043–1052.

[22] C. Thompson, J. White, B. Dougherty, and D. C. Schmidt, "Optimizing Mobile Application Performance with Model-Driven Engineering," in *Proceedings of the 7th IFIP WG 10.2 International Workshop on Software Technologies for Embedded and Ubiquitous Systems*, ser. SEUS '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 36–46.

[23] G. Rose, "Mobile Phones as Traffic Probes: Practices, Prospects and Issues," *Transport Reviews*, vol. 26, no. 3, pp. 275–291, 2006.

[24] P. Leijdekkers and V. Gay, "Personal Heart Monitoring and Rehabilitation System using Smart Phones," in *Proceedings of the International Conference on Mobile Business*. Washington, DC, USA: IEEE Computer Society, 2006.

[25] H. Giese, B. Rumpe, B. Schätz, and J. Sztipanovits, Eds., *Science and Engineering of Cyber-Physical Systems (Dagstuhl Seminar 11441)*, ser. Dagstuhl Reports. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011, vol. 1(11).

[26] M. Heemels and G. Muller, *Boderc: Model-Based Design of High-tech Systems*, 2nd ed. Den Dolech 2, Eindhoven, The Netherlands: Embedded Systems Institute, March 2007.

[27] P. Fritzson and V. Engelson, "Modelica - A Unified Object-Oriented Language for System Modelling and Simulation," in *ECCOP '98: Proceedings of the 12th European Conference on Object-Oriented Programming*. Springer-Verlag, 1998, pp. 67–90. [Online]. Available: http://www.modelica.org/documents/ModelicaSpec32.pdf

[28] G. Nicolescu, H. Boucheneb, L. Gheorghe, and F. Bouchhima, "Methodology for Efficient Design of Continuous/Discrete-Events Co-Simulation Tools," in *High Level Simulation Languages and Applications*, J. Anderson and R. Huntsinger, Eds. San Diego, CA: SCS, 2007, pp. 172–179.

[29] J. Eker, J. Janneck, E. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong, "Taming Heterogeneity – the Ptolemy Approach," *Proc. of the IEEE*, vol. 91, no. 1, pp. 127–144, January 2003.

[30] C. G. Cassandras, *Analysis and Design of Hybrid Systems: a Proceedings Volume from the 2nd IFAC Conference*. Elsevier, Jun. 2006.

[31] M. Renger, G. L. Kolfschoten, and G.-J. Vreede, "Challenges in Collaborative Modeling: A Literature Review," in *Advances in Enterprise Engineering I*, ser. Lecture Notes in Business Information Processing, J. L. G. Dietz, A. Albani, J. Barjis, W. Aalst, J. Mylopoulos, M. Rosemann, M. J. Shaw, and C. Szyperski, Eds. Springer Berlin Heidelberg, 2008, vol. 10, pp. 61–77.