Software Engineering for Smart Cyber-Physical Systems:
Challenges and Promising Solutions

Tomas Bures[1], Bradley Schmerl[2], Danny Weyns[3], Eduardo Tovar[4] (workshop organizers)
Eric Boden[5], Thomas Gabor[6], Ilias Gerostathopoulos[7], Pragya Gupta[8], Eunsuk Kang[9], Alessia Knauss[10], Pankesh Patel[11], Awais Rashid[12], Ivan Ruchkin[3], Roykrong Sukkerd[3], Christos Tsigkanos[13] (contributing participants)

[1]Charles University Prague, [2]Katholieke Universiteit Leuven & Linnaeus University, [3]Carnegie Mellon University, [4]CISTER-ISEP, [5]University of Paderborn, [6]LMU München, [7]TU München, [8]fortiss, [9]Massachusetts Institute of Technology, [10]Chalmers University of Technology, [11]ABB Corporate Research, [12]Lancaster University, [13]Politecnico di Milano

bures@d3s.mff.cuni.cz, danny.weyns@kuleaven.be, schmerl@cs.cmu.edu, emt@isep.ipp.pt, eric.boden@uni-paderborn.de, thomas.gabor@ifi.lmu.de, ilias.gerostathopoulos@tum.de, gupta@fortiss.org, eunsuk.kang@berkeley.edu, alessia.knauss@chalmers.se, pankesh.patel@in.abb.com, a.rashid@lancaster.ac.uk, iruchkin@cs.cmu.edu, rsukkerd@cs.cmu.edu, christos.tsigkanos@polimi.it

Abstract

Smart Cyber-Physical Systems (sCPS) are modern CPS systems that are engineered to seamlessly integrate a large number of computation and physical components; they need to control entities in their environment in a smart and collective way to achieve a high degree of effectiveness and efficiency. At the same time, these systems are supposed to be safe and secure, deal with environment dynamicity and uncertainty, cope with external threats, and optimize their behavior to achieve the best possible outcome. This "smartness" typically stems from highly cooperative behavior, self-awareness, self-adaptation, and self-optimization. Most of the "smartness" is implemented in software, which makes the software one of the most complex and most critical constituents of sCPS. As the specifics of sCPS render traditional software engineering approaches not directly applicable, new and innovative approaches to software engineering of sCPS need to be sought. This paper reports on the results of the Second International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS 2016), which specifically focuses on challenges and promising solutions in the area of software engineering for sCPS.

Keywords: software engineering, cyber-physical systems

Introduction

Cyber-Physical Systems (CPS) are "engineered systems that are built from, and depend upon, the seamless integration of computational and physical components" [1]. With the proliferation of smart embedded and mobile devices, CPS are becoming large-scale pervasive systems, which combine various data sources to control real-world ecosystems (e.g., intelligent traffic control) [2,3,4]. Modern CPS have to deal effectively with environment dynamicity, control their emergent behavior, be scalable and tolerant to threats, hence CPS have to be smart (sCPS).

Compared to traditional embedded systems, i.e. hardware-intensive systems with well-defined interfaces and boundaries delivering specific services to their end-users often under stringent reliability and safety requirements, sCPS are more interconnected and more dependent on software for their operation. Car infotainment products, for instance, have reached the size of multiple millions lines of code. This complexity alone already calls for systematic software engineering (SE) models, methods, and processes for building such sCPS. At the same time, sCPS feature a number of specifics that render traditional SE approaches (e.g. component-based modularization, simulation-based validation) not directly applicable. These specifics include the blurring boundaries between hardware and software, large scale and complexity, the role of end-users, inherent uncertainty, open-endedness, locality, etc. What is needed are innovative approaches that jointly reflect and address the specifics of such systems.

SEsCPS Workshop

The SEsCPS workshop series[1], part of ICSE, aims at addressing this gap in software engineering for sCPS by looking at the specifics of sCPS, along with opportunities and challenges tied to them. The workshop brings together academics, practitioners, and trainers from several disciplines with the overall objectives: (i) to increase the understanding of problems of Software Engineering (SE) for sCPS, (ii) to study the underlying foundational principles for engineering sCPS, and (iii) to identify and define promising SE solutions for engineering of sCPS.

In this report, we summarize the findings of the 2nd edition of the workshop, held on May 16th, 2016 in Austin, Texas in conjunction with ICSE 2016.

Workshop Structure

Based on the interests shown by participants at the previous edition of the workshop [5] and research interests shown at related venues, the special themes of SEsCPS'16 were: (1) alignment of disciplines for engineering sCPS, (2) uncertainty and human factors, and (3) reference problems. Around these themes, the workshop strived to build understanding of sCPS and provide a basis for holistically addressing the SE challenges brought by sCPS.

The second edition of the workshop attracted 17 submissions, out of which 7 were accepted as full papers and 2 as position and future-trends papers. In total, around 25 participants attended the workshop. The workshop started with a keynote. The rest of the morning was devoted to presentations of accepted papers, grouped in three themes as overviewed in the next section. The whole af-

[1] http://d3s.mff.cuni.cz/conferences/sescps2016

ternoon of the workshop was devoted to discussion in breakout groups, where participants discussed focused topics of SE for smart CPS. A plenary report session concluded the workshop.

Keynote

The keynote was delivered by Eric Bodden (University of Paderborn, Germany), who focused in his talk on the important subject of security of sCPS. He highlighted that the high degree of connectivity of sCPS renders them vulnerable to a whole range of new security attacks. He argued throughout the talk that sCPS so far are insecure for two main reasons. First, because of a lack of integrated perspective on the inherent complexity of security mechanisms that take into account both software and hardware vulnerabilities. Second, because the companies behind the inception and engineering of innovative sCPS currently lack competence in software engineering, since they were never software companies, but car makers, home and industrial electronics manufacturers, and telecom equipment and services providers.

Workshop Themes

The workshop presentations provided a cross-cutting view of the software engineering challenges related to sCPS and potential approaches to address the challenges. The presentations were organized into the three themes overviewed below.

Formal Modeling and Planning

The first theme of the workshop was concerned with formal modelling and planning of sCPS. sCPS are typically at least partly including critical infrastructures (e.g. smart mobility, smart grids, etc.). As such, formal modeling techniques are crucial to derive guarantees pertaining to sCPS and provide abstractions needed to investigate the potential behavior of sCPS. Being subject to a large degree of uncertainty, sCPS typically combine guarantees with graceful degradation and planning to control guarantees and optimality of their behavior. This brings important research questions related to how to model sCPS (including abstractions and semantics) and how to tie these models to guarantees and planning.

These topics were targeted by three presentations. Christos Tsigkanos argued that Building Information Models, the de facto standard for specifying complex information about building infrastructures, can also be extended for the specification of cyber-physical aspects. By providing formal static and dynamic semantics of the cyber-physical spaces they induce in terms of topological concepts it is possible to support many forms of advanced analyses typically performed in software engineering. Pragya Kirti Gupta presented a formal approach for calculating guarantees for the objective of maximizing availability and survivability using graceful degradation. The model represented formal specification of the physical entities and the constraints under which these physical entities must operate. The case study of a microgrid operating in an island mode was analyzed. The work showcased the use of model-based development approach in controlling and managing CPS at large. Roykrong Sukkerd proposed a multi-scale temporal planning approach for state-based planning to control the complexity of employing both the required fidelity of time discretization of a problem domain, as well as a long planning horizon that enables planning to yield closer-to-optimal solutions. We use a discretization scheme such that the size of time lattice increases with time, leveraging the fact that prediction uncertainty of the environment increases with time and thus there is less precision of time of occurrences of far-future events.

Safety and Security

The second theme discussed at the workshop was safety and security. As sCPS will be more and more immersed into everyday life, the requirements on their safe operation, one that does not jeopardize lives, will only increase. At the same time, solving the security challenges imposed by sCPS, related to both old and new attacks, will be a major issue in wide-spread adoption of such systems.

These topics were targeted by four presentations. Eunsuk Kang presented a design-time approach for automatically generating security attacks on a CPS using the Alloy formal modeling and analysis tool. He described an application of this approach to discover real attacks on a fully functional water treatment testbed at the Singapore University of Technology and Design (SUTD). Charles Walter presented a wearable adaptation management application to prevent attackers from stealing private information by eavesdropping on Bluetooth communication. Since the targeted wearables connect with a phone as the base station, a phone app was created that intelligently adapts the level of security based on data from every connected device and the phone. This approach opens the possibility for learning user needs for security and providing a mechanism to update policies as the user interacts with the security awareness application. Alessia Knauss presented first results from their investigation on the state-of-the-art and future trends in testing critical CPS on the example of active safety systems. The results were based on four focus groups with 11 practitioners from Sweden and differentiate between the original equipment manufacturer (OEM) and supplier point of view. The results underline the current support of major testing processes, however, there is a clear need to develop support for the testing of more complex scenarios in realistic settings. Furthermore, the degree of automation of testing needs to be increased to support repeatability and more effective test resource usage. Awais Rashid asked if the human-in-the-loop is indeed the weak link regarding security incidents, highlighting the role of *latent design conditions* in impacting operators' perceptions of adversarial behavior in cyber-physical systems. He then discussed the characteristics of smart CPSs—namely, their dynamically aggregated nature, emergent behavior and multi-stakeholder environments—that make it particularly challenging to address the impact of their inherently *emergent design* on operators' perceptions of security events.

Frameworks

Since sCPS are large complex systems with large codebases, adhering to certain rules or following certain guidelines can facilitate their software development. Such rules or guidelines typically constrain the software developers in their creative process; however, the conventions that are followed produce understandable, maintainable, modular, and extensible software. Software development rules and guidelines are typically grouped together into component models, patterns, architectural styles and frameworks.

This topic was targeted by the last two presentations. Pankesh Patel proposed a development framework that segregates CPS development concerns, provides a set of modelling languages to specify them, and integrates automation techniques to parse these modelling languages. Thomas Gabor presented a taxonomy of challenges to be faced when integrating autonomous decision making into the development cycle of future sCPS, allowing to analyze changes to be made to current development processes.

Open Research Topics

The whole afternoon of the workshop was allocated to breakout groups that focused on selected topics from the morning presentations. In total, there were four groups, each focusing on one of the topics selected for discussion: "Software Engineering Process for sCPS", "Emergent Designs", "Consequences of Smartness and Security", "Should Software Provide Security to Control?" In the rest of the section, we report on the findings of each breakout group in turn.

Emergent Designs

The first breakout group focused on the research trends in sCPS. sCPS points to complex systems that not only control the physical entities, but also coordinate various business processes [7]. Architectural design strategies have been suggested to create platforms for sCPS [8]. This has given a new impetus for research in machine learning, architectural design, smart user interface, recommender systems etc. According to the report by Gartner, emerging technologies in 2016 such as machine learning, software-defined security, IoT platforms, micro data centers etc. are at the peak of the hype cycle [9].

The group discussion was around the clarification of the terms "cyber-physical" and "smartness". In order to define these terms, key topics that emerged were uncertainty and the representation of the environment.

Uncertainty or dealing with what is unknown largely depends on the representation of all the factors, which affect the functioning of the system. Therefore, it is important to know the functionalities, structure and the workflow of the system internal to it. In other words, when using a model-based approach, the model must capture the data flows, functionalities etc. Requirements must be explicitly defined. It was agreed upon that even if the objectives and the functionalities were clearly defined, the environment in which the system is running is equally important. The changes within the system in turn affect the environment. Challenges of capturing the unknowns of the environment that affect the system largely remain open research. Therefore, it totally justifies that the research trend we see in emerging technologies that focuses on maximizing the data provisioning and knowledge extraction algorithms.

It was also discussed that the clarification of "smartness" could only be possible when the expectations from complex system such as energy system, autonomous vehicle, and smart industry are clearly defined. In other words, the requirements and specification of the complex system must be defined properly and completely. The detection of conflicts amongst the requirements of complex systems that operate simultaneously remains a research challenge. For example, charging the autonomous vehicle using Electric Vehicle (EV) charging stations in a smart city requires clarity of the requirements related to EV charging stations, autonomous vehicles energy requirement, layout and design of the city that provides energy to the charging stations.

Software engineering practices can help in analyzing requirements, defining architectures that allow interoperability and checking for conflicts between complex systems.

In this context, challenges can be categorized as following;

- Adaptation and evolution in sCPS.
- "Uncertainty" and representation of environment.

sCPS are expected to adapt in certain situations. These adaptations might involve changes in the structure or just fine-tuning certain parameters. The major challenges and open research areas pertaining to adaptation are:

1. How to identify the missing functionalities in the continuously evolving system?
2. How the workflow will evolve if adaptation is introduced by making structural changes?
3. How to design the evaluation platforms that analyze and compare the initial and adapted design?
4. What are the metrics required to compare the initial design with the evolved design?
5. How to recognize if a change is an opportunity to adapt or is loss of functionality?

Consequences of Smartness and Security

The second breakout group focused on the interplay of smartness and security, based on the observation that smartness typically involves more complex functionality and more complex interactions, which, in turn, increases the potential attack surface and thus has a negative effect on security.

Connecting to the keynote by Eric Bodden, the group started by clarifying the terms and scope of smartness and sCPS. The group identified smartness as: "being connected and able to act with partial autonomy based on own decision making while cooperating and coordinating with others." sCPS were then seen as CPS exhibiting a significant degree of smartness, which implies that sCPS (as the collection of multiple distributed and partially autonomous cooperating nodes) take decisions on their own and exhibit self-* properties.

The ability to take own decisions is not binary. It rather makes a continuous scale where some class of decisions can be taken autonomously, whereas another class of decisions may require approval or incentive by a human stakeholder. Consequently, even the "smartness" of sCPS is to be seen as a continuum (as opposed to classifying some CPS as "smart" and some others as "dumb").

Having established the common understanding, the group focused on security related issues connected with smartness. Apart from the problem of increasing the potential attack surface, an important security problem stems from the sCPS's ability to make autonomous decisions. This makes it possible to forge attacks that impact a system's sensing and self-awareness, thus forcing it into making a wrong decision. This makes it possible to impact the system even without breaching into its control functions. Similar problems happen even in systems controlled by human operators (as discussed during the presentation of Awais Rashid), however sCPS are more vulnerable due to their inability to reflect on potential decisions from the perspective of the "common sense."

The smartness does not introduce only problems for security, but can be seen also as an opportunity to implement mechanisms strengthening the security. As smartness allows the system to introspect and reason about its environment (including potential security threats), the system can protect itself. An existing example of this is the moving target defense strategy. Also, thanks to its high degree of autonomy, the system requires fewer inputs from human operators. The system can essentially control its behavior

and system boundary from inside. This reduces the possibility of attacks compromising the channel and the credentials via which the operator accesses the system.

Generalizing these observations, it is expected that sCPS may enable new types of attack patterns that we do not currently know. Nevertheless, they may be somehow latent in existing attacks, thus forming a kind of "unknown knowns" [6]. The smartness, however, may make the system more resilient. Here, the pertinent question is how we could exploit the smartness in strengthening the security. The role of software engineering in this is critical, as it should make it possible to reason about security and smartness in a systematic way. Disciplined software engineering methods needs to propel the ability to reason about the system as a whole and seamlessly incorporate defense strategies specifically tailored towards the vulnerabilities of sCPS.

An interesting example was drawn from the keynote. In 2011, there was a hack attempt targeting Lockheed Martin. Instrumental in the attack was the ability of the attacker to compute access codes generated by SecurID tokens used in two-factor authentication. In order to compute the codes, the attackers previously compromised the servers of RSA Security (the manufacturer of the SecurID tokens) and obtained seeds used by the tokens to generate the access codes. Among others, this story shows that under certain circumstances the security-related system boundary of CPS can be effectively larger than originally assumed. In this case, due to the chain of trust, the security-related system boundary of Lockheed Martin in some sense also included the servers of RSA Security. This pattern can be found in many other systems in nowadays interconnected world. Here, the smartness featured by sCPS could actually make it possible to shrink the system boundary back to units of manageable size, which would be able to autonomously reason about actions performed with them and protect themselves against an attack.

In addition to the topics discussed above, the group further identified a number of important research questions / challenges listed (in no intentional order) below:

- How does one measure and quantify the security?
- How to figure out that the system has been compromised and how to react to it? Especially by reactions that cannot be easily guessed by an attacker.
- How do we know what the realistic settings are if those systems do not exist yet? Here, a big problem is the current lack of large sCPS testbeds.
- How to coordinate security behavior of multiple sCPS and how to cope with the potential lack of the global perspective? Obviously, the smarter the sCPS are, the more difficult this is. The security should be a joint effort of multiple systems, but adaptation of one system may easily compromise guarantees and assumptions of other systems. This leads to an important issue of compositionality of security guarantees.

Should Software Provide Security to Control?

The third breakout group focused on a question phrased in perhaps an unusual way, "should software provide security to control?"

The goal of the phrasing was to challenge the assumption that security is a passive quality designed up-front for the whole system (which includes control algorithms and other software). As the question suggests, some parts of the system (namely, software) can play an active role in providing intelligent security, particularly in systems built upon control mechanisms.

Taken to the security context of sCPS, the dichotomy between "software" and "control" becomes unsuitable for two reasons:

1. There is no clear separation between "control" and "software". If we understand control narrowly (in a control theoretical/engineering sense), it has been increasingly implemented in software over the last decade. If we understand control broadly, a lot of software has some control function in it.

2. Security is a system-wide quality—not a particular functional requirement. We cannot say that any specific component provides security for the system. Instead, security depends on how parts of the system come together and interact with a particular environment. Even when some components have active security functions (e.g., moving target defense), security reasoning and evaluation needs to take the whole system into account.

Let us then understand the interplay between control, security, and software in sCPS better. Consider a system that uses classic control as a functional skeleton that interacts with multiple physical processes. On top of that, coordination software weaves together these control loops to ensure the satisfaction of system qualities, including security. This group's discussion focused on how to improve engineering of such secure sCPS. In contrast, the last subsection discusses engineering for sCPS in general, and the previous subsection addresses the topic of smart decision-making for security. In the remainder of this subsection, we explore how security engineering of sCPS needs to differ from security engineering of "pure software" systems and classic control systems.

Our starting question hints that classic control is often not prepared to deal with security challenges in diverse and rapidly changing environments. In part, this deficiency is due to *control's focus on overcoming random failures* (caused by noise and natural processes). Security engineering, in contrast, needs to deal with intentional intrusions. Such intrusions can upset classic control that does not expect input errors to deviate from the normal distributions. This weakness gives rise to difficult-to-prevent sensor spoofing attacks, such as the one carried out on Jeep's GPS [7].

Another challenge in sCPS, both for classic control and cybersecurity, is the *increased scope and scale of systems*. Now systems include not only software and individual physical processes, but also groups of interrelated physical processes, humans, and other smart systems. This diverse context creates ample opportunities for lateral movement of the attacker, who can create attack chains switching between compromising software (e.g., with an exploit), compromising a physical process (e.g., with sensor spoofing), to compromising a human (e.g., with phishing).

Another difference for security in sCPS is *smartness* – the system's ability to learn and respond intelligently in new contexts. We distinguished two degrees of smartness. The first-degree smartness is existence of intelligent end-user devices that coordinate with each other. This smartness creates many new attack surfaces, and is a great security challenge. The second-degree smartness means that the system deliberately learns and acts on its smart end-user devices. Often, this smartness is considered benefi-

cial to security since it makes intelligent decisions to maintain security at run time (as discussed in the previous section). However, second-degree smartness comes with challenges of its own. Automated learning can be misled by attackers into overfitting to a specific solution, or disrupted to prevent learning. In some way, it seems that no matter how much smartness is added, security engineers will always be one step behind attackers: smartness of degree *n-1* introduces vulnerabilities that are patched up by smartness of degree *n*, but that smartness comes with its own vulnerabilities, requiring yet another level, and so on.

The group discussed two examples of learning attacks. The first attack is an attack on modes in learning. Suppose a system has several behavioral models, pertaining to criticality or safety. Each mode requires separate learning of acceptable behaviors. Such a system can be disrupted if an attacker finds a way to trigger mode switches, thus preventing each mode's learning to be reset and never accumulate enough continuous data to behave appropriately. The second attack is an attack on historical records. In this attack, the attacker would compromise the historical record of the system's operation, or the component that creates that record. After that, no further escalation of privilege may be needed: the attacker can lead the system into a desired behavior by tweaking the historical records, which make up the training set for the system's learning algorithms.

To respond to the above challenges, the group discussed avenues for future research on security engineering for sCPS. Clearly, the existing cybersecurity approaches are reliant on relative isolation of software from physical processes and humans. To bridge this gap in sCPS, one suggestion was to increase the set of existing security tactics by drawing inspiration from other fields. For instance, one can use tactics from reliability engineering such as redundancy and aggregation of diverse signals. From physical security, we can borrow separation of system's elements and hardening. Tactics from classic cybersecurity (such as diversity and encryption) can be upgraded to the sCPS context by considering their effects in a broader system context.

Another way to address the discussed challenges is to change the security mindset for sCPS. Instead of trying to prevent any attack, security engineers can focus on how to best respond to attacks via adaptation. The adaptive defenses can continuously stratify the system into critical, controllable, and uncontrollable parts. With this information, the system can guide its responses appropriately without spending resources on trying to recover a part where the control has been lost.

Finally, the group discussed implications for practical engineering:

- Plugging in well-tested solutions might not work due to rapid change in context.
- Supply chains, often providing low-level devices and functions, need a systematic way to be scrutinized for the broadened notion of security.
- Security analysis for sCPS needs to become consistent – more repeatable with more predictable results, even by diverse groups of experts. Attack modeling and simulation may become one such consistent approach.
- Instead of security reviews by experts, hacking competitions have been suggested as a method of evaluating and improving a system's security. The system's builder would announce a prize for compromising a system, and a number of teams would try to break the system's security.

To summarize this section, sCPS brings several challenges to secure engineering of control software: intentional deviations in input errors, increased scope of systems, and attacks on learning. The discussion indicates that security for sCPS is a broad and uncertain quality, requiring analysis and intelligent response at run time. The group suggested borrowing tactics from other fields and giving adaptive security a richer way to stratify the system's context. This discussion has certainly not been exhaustive, and more engineering challenges for secure sCPS will shape the landscape of future research.

Software Engineering Process for sCPS

The fourth breakout group focused on software engineering processes for the development of sCPS. One common theme in the contributions of the workshop as well as the respective research field includes the application of well-established principles from software engineering to enhance the non-trivial research field of cyber-physical systems. Furthermore, as the theme of this workshop focused on smart cyber-physical systems, the group discussion raised the question, if and why smartness of CPS even affects the engineering process: Many models used in software development are general enough and do not require any assumptions about the software systems they are used to build. In order to recognize the ways in which sCPS may differ from more traditional CPS, this group concluded that a precise definition of smartness is detrimental to further analysis.

As an impromptu solution, the group defined smart systems as systems that: (1) can adapt to a variety of situations instead of being built for a fixed scenario, (2) gathers data about its environment and its own state throughout its operation, and (3) uses that data to improve its behavior at some future point in its life-cycle, which may be a future version of the system deployed by the system developer or even be an adapted version of the system generated at runtime via some form of artificial intelligence. The group noted that many such systems (autonomous vehicles, smart cities, personalized healthcare appliances, e.g.) are currently gaining attention and increasingly created. These systems would greatly benefit from insights regarding systematic development approaches for smartness.

From this rather simple definition, the group managed to derive a few ways in which smartness poses a challenge to current software development processes. In one way, smart behavior relies on the interaction of the system with its surroundings, enabling the smart system to not only perceive and react on the environmental information but also deliberately plan changes in the environment. The need to interleave the development of hardware and software, which is already felt in all CPS, is thus aggravated in sCPS. It thus shows that smartness introduces a whole range of new cross-cutting concerns into the development process, out of which the breakout group identified three major groups of development aspects that seem to be the most essential missing links at the moment:

- First, even though systems are getting ever smarter, they are going to make mistakes as well, which needs to be detected. As smart systems are deployed for tasks that are

too complicated or too work-intensive to be performed by humans, their errors may not be easily detectable for a human supervisor. It is necessary to employ special techniques to ensure the predictability and liability of a smart system's behavior. An important factor to generate trust in an autonomous machine's actions is transparency: The system needs to find the right decision and be able to explain that decision to humans.

- Second, as a sCPS will adapt and change its behavior at runtime, such systems should be debug-able. If a human supervisor detects a problem with the system's behavior, techniques are needed that would allow to fix the problem, even with all the difficulty of tracing it back to its origin.
- And third, this whole process of error detection and correction itself needs to be reliable enough so that the safety of the sCPS can be certified. Certification of autonomous systems, however, appears to be an area still largely untouched.

Having discussed concerns that cross-cut over several components of the software architecture, adding smartness to CPS also affects different phases of traditional development processes:

- Requirements need to be defined in a more general manner allowing to accommodate for openness of the system and the uncertainty it is dealing with during operation. More abstract requirement specification, however, can yield many different instances of problems within the space of parameters spanned by the requirements.
- The system design should accommodate for a variety of configurations targeting the different situations that the system will encounter during its life time—possibly even unforeseen situations.
- Testing of sCPS during design time can only provide very limited guarantees towards actual run time behavior when the system is allowed to reconfigure itself during run time.

In general, many of the engineering steps can be performed during system design for non-autonomous systems. However, these activities are going to be shifted to the operations phase and thus must be executed (at least in parts) during run time. Engineering such systems becomes a "perpetual process" that unifies tool-supported development and evolution with automatic runtime adaptation [8]. Realising this shift will require new models on how to distribute tasks along the development cycle and new tools to enable tasks like reconfiguration, configuration testing, and quality assurance to be split between design time and run time. Tools that help the system to handle complexity, deal with uncertainty, and take advantage of runtime data are needed. When developing new tools for sCPS, their integration into the life-cycle must adhere to a plug-and-play paradigm, i.e., for every reconfiguration performed on the system, there must be a clear (and ideally quick) way to apply the tool to new parts of the system to ensure that the results of the tool can be preserved for the system as a whole.

Finally, the discussion concluded with a few questions that have been considered most urgent to answer by the group members and may provide fruitful directions for future research of sCPS:

- How can "smartness" be defined and how exactly does it relate to concepts like "self-adaptation" or "artificial intelligence"?
- Traditional software engineering represents rigid processes: Is it helpful to consider this rigidness a goal for software engineering even for smart systems that might be achieved after gaining more experience on how to build these? Or would it be more helpful to abandon the view of system building as an exact discipline of engineering when it comes to smart systems?
- Adding "smartness" is a very general approach to increase the system performance for a variety of systems: Should solutions for different domains be sought of on the same level of generality? Or does the controllability of smart system design only arise when considering more specific domains?

Acknowledgements

The SEsCPS 2016 workshop is a collective endeavor, as such, the authors would like to express their gratitude to those who have participated in the organization of this workshop. This comprises in particular the ICSE 2016 Workshops Co-Chairs, in particular Marija Mikic and Mauro Pezzè, and the SEsCPS 2016 Program Committee comprised of Paris Avgeriou, Steffen Becker, Nelly Bencomo, Johann Bourcier, Radu Calinescu, Jan Carlson, Sagar Chaki, Ivica Crnkovic, Nicolas D'Ippolito, Rogério de Lemos, Dionisio De Niz, Antonio Filieri, Ilias Gerostathopoulos, Carlo Ghezzi, Holger Giese, Rodolfo Haber, Gabor Karsai, Mark Klein, Filip Krikava, Martina Maggio, Henry Muccini, Maurizio Murroni, Geoffrey Nelissen, Gurulingesh Raravi, Wolfgang Renz, Bernhard Schaetz, Ina Schieferdecker, Lionel Seinturier, Vitor E. Silva Souza, Bedir Tekinerdogan, Petr Tuma, and Steffen Zschaler.

References

[1] NSF, Cyber Physical Systems, NSF 15-541 http://www.nsf.gov/pubs/2015/nsf15541/nsf15541.pdf

[2] B. K. Kim and P. R. Kumar, "Cyber–Physical Systems: A Perspective at the Centennial", Proceedings of the IEEE, vol. 100, no. Special Centennial, 2012.

[3] E. A. Lee, "Cyber Physical Systems: Design Challenges", 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing, 2008.

[4] L. Sha, S. Gopalakrishnan, X. Liu, and Q. Wang, "Cyber-Physical Systems: A New Frontier," IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing, 2008.

[5] T. Bures, D. Weyns, S. Biffl, M. Daun, T. Gabor, D. Garlan, I. Gerostathopoulos, C. Julien, F. Krikava, R. Mordinyi, "Software Engineering for Smart Cyber-Physical Systems – Towards a Research Agenda," Software Engineering Notes, November 2015.

[6] A. Rashid, S. Asad A. Naqvi, R. Ramdhany, M. Edwards, R. Chitchyan, and M. A. Babar. Discovering \unknown known" security requirements. In Proceedings of the 38th International Conference on Software Engineering, 2016.

[7] Andy Greenberg. "Hackers Remotely Kill a Jeep on the Highway—With Me in It." In *WIRED*, July 21, 2015.

[8] D. Weyns, N. Bencomo, R. Calinescu, J. Camara, C. Ghezzi, V. Grassi, L. Grunske, P. Inverardi, J. Jezequel, S. Malek, R. Mirandola, M. Mori, and G. Tamburrelli. "Perpetual Assurances in Self-adaptive Systems." In Software Engineering for Self-Adaptive Systems III. Lecture Notes in Computer Science, Springer, 2017.