

Engineering a Cyber-Physical Intersection Management – An Experience Report

Florian Wessling, Stefan Gries^(✉), Julius Ollesch, Marc Hesenius,
and Volker Gruhn

paluno - The Ruhr Institute for Software Technology, University of Duisburg-Essen,
Schützenbahn 70, 45127 Essen, Germany
{florian.wessling,stefan.gries,julius.ollesch,marc.hesenius,
volker.gruhn}@paluno.uni-due.de
<http://se.paluno.uni-due.de>

Abstract. The engineering of cyber-physical systems (CPS) imposes a huge challenge for today’s software engineering processes. Not only are CPS very closely related to real objects and processes, also their internal structures are more heterogeneous than classical information systems. In this experience report, we account on a prototypical implementation for an intersection management system on the basis of physical models in the form of robotic cars. The steps to implement the working physical prototype are described. Lessons learned during the implementation are presented and observations compared against known software processes. The insights gained are consolidated into the novel *Double Twin Peaks* model. The latter extends the current software engineering viewpoints, specifically taking CPS considerations into account.

Keywords: Software engineering · Cyber-physical system · CPS · Requirements engineering · Twin Peaks · Modeling · Experience report · Agile development · Software process model

1 Introduction

Cyber-physical systems (CPS) are an emerging topic for research and enable digital innovation in domains such as energy, health and transportation [13]. Equipped with computing power, networking and the ability to sense and actuate real world processes, CPS enable ambient intelligence to conduct process control. Furthermore, CPS may consist of heterogeneous components unknown at development time, thus allowing dynamic extension at runtime allowing greater adaptability and the ability to cope with heterogeneous infrastructures. Being defined as systems at the crossroad between physical processes and information processing [12], CPS are vital for ambient intelligence.

CPS are key in the digital transformation – a development that brings the software engineering and traditional engineering domains closer together and creates new and interesting technical challenges and opportunities [14]. However,

an often overlooked problem is the orchestration of experts to create CPS – i.e. the engineering process itself. To date, to the best of our knowledge there are no comprehensive accounts on the engineering methods for CPS. Thus, there is an urgent need to explore similarities and differences between information system engineering and engineering of CPS.

Road-going vehicles are currently undergoing a transformation to gradually become networked and autonomous [7]. The anticipated impacts are profound: greater individual productivity, less accidents and killings and new emerging business models to name only a few. Notably, a directly affected aspect of this transformation is traffic regulation.

Today, traffic signals and road signs are designed to be human-readable, but in the near future there is potential to optimize traffic flow by directly communicating with connected self-driving cars. While there are research groups focussing on analyzing and improving intersections and traffic flows in general, to the best of our knowledge, this is the first approach with a focus on the engineering of such an intelligent system.

The example of an intersection management system seems appropriate to study how CPS evolve as its engineering does not only concern information systems but also mechanical components, sensors and actuators. To this point no general engineering process has emerged for CPS. Hence, we chose to investigate and observe the process of building a physical prototype using the example of an intersection with the aim to derive good practices that may be generalized in future work.

The paper is organized as follows: in Sect. 2 we will review related work on intersection simulation, physical prototyping and software engineering processes. Sect. 3 presents the project approach and describes the work done in the respective sprints. Section 4 summarizes the results of the project in terms of intersection management and technical challenges while Sect. 5 describes the software engineering process which emerged from the development of the prototypical CPS. Eventually Sect. 6 provides an outlook in terms of future research directions.

2 Related Work

2.1 Software Engineering Process

Traditional software engineering processes are not directly applicable to cyber-physical systems due to the specific characteristics of CPS such as a close interdependency between hard- and software, uncertainty during operation, as well as the large scale, complexity and distribution of infrastructures [1, 5]. As CPS lie at the intersection of multiple disciplines such as mechanical engineering, electrical engineering, control engineering, software engineering and physics, the CPS engineering process is multidisciplinary as well [1, 6, 10].

Al-Jaroodi et al. [1] give an broad overview of the software engineering challenges imposed by CPS. One of their findings is that the complexity strongly depends on the domain of the cyber-physical systems under development. While mobility and power limitation might be issues for CPS in the automotive domain

this restriction is different for CPS developed in a smart home context. According to Al-Jaroodi et al. all software engineering phases such as analysis, design, implementation and testing need to be reconsidered and adapted when developing CPS [1]. Particularly during the analysis phase models and tools are required that enable a coherent specification while capturing CPS characteristics. For example Bures et al. consider the idea to use software architecture models that are extended with knowledge and insights “such as electro-mechanical elements, physical constraints and laws” from other areas [5].

The view from a software vendor is presented by Röchardt and Bräuchle [18]. Their experiences support the initial assumption of this paper that the interdependency between hard- and software has a huge impact. The authors explain that the business model is influenced as well: “experiences with enterprise systems can be extrapolated and transformed into a new model of system operations, where product and service merge to form one common business model.” [18].

Autonomous driving is currently gaining public interest and CPS are enabling this trend. Therefore we have chosen the domain of automotive traffic coordination and aim at creating a small-scale physical intersection in order to examine the software engineering process for CPS.

2.2 Simulating Intersections

There are several approaches working towards the future of intelligent transportation systems in which vehicles cross intersections autonomously.

A first step towards making intersections more efficient is analyzing the dilemma zone problem, which refers to the area in front of an intersection that is approached during the yellow light phase and the driver being indecisive about stopping or crossing the intersection. Petnga and Austin [17] describe this dilemma as a set of conditions that represent an unsafe state and present a simulation framework for implementing resolution algorithms. The authors conclude that for achieving a successful coordination it is necessary to consider cars and traffic lights simultaneously, i.e. both spatial and temporal data is required in order to prevent the system from reaching an unsafe state.

In a recent work from MIT the authors Tachet et al. examine slot-based systems known from aerial traffic coordination and present a framework to analyze the performance of different algorithms for a slot-based intersection for vehicles [20]. The common way of coordinating vehicles are traffic lights which grant access to an intersection area (i.e., the shared resource) exclusively to one of the traffic directions. In contrast to this approach, slot-based systems consider the trajectory of multiple vehicles and prevent collisions by coordinating the time slot in which the intersection can be crossed safely (simultaneously for multiple traffic directions). Their work shows that by using a slot-based intersection the capacity of an intersection can be doubled and delays significantly reduced.

Azimi et al. focus on Vehicle to Vehicle (V2V) communication in order to coordinate the crossing of an intersection [3]. Their algorithm segments the intersection into a grid of 4×4 cells. Each vehicle calculates possible collisions with

other vehicles based on their desired trajectory and its respective occupied cells during the crossing. This communication process is triggered when approaching the intersection. The authors experiment with different algorithms (“intersection protocols”) and compare the trip time and delay caused by crossing the intersection. Their results show that by avoiding single colliding cells on vehicle trajectories the delay caused by common traffic lights can be reduced from 48% up to 85% due to lower waiting time and a more fine-grained planning.

Wuthishuwong and Traechtler use a Vehicle to Infrastructure (V2I) approach in which a centralized system plans and coordinates the trajectories of vehicles crossing an intersection [21]. Based on discretization of the vehicle’s two-dimensional trajectory and considering time as the third dimension the authors employ Dynamic Programming to calculate a collision-free route for each vehicle. Compared to the aforementioned approach Wuthishuwong and Traechtler achieve an even more fine-grained trajectory. Although the authors did not carry out exhaustive experiments they state that their approach reduced delay and supported an continuous flow of vehicles crossing an prototypical simulated intersection.

2.3 Simulations and Physical Prototypes

The examples mentioned in Sect. 2.2 are all software-based and simulate vehicle trajectories without any connection to physical devices.

The work by Paczesny et al. studies the link between simulation and prototyping of cyber-physical systems [16]. By providing a middleware combining aspects from both areas, it is possible during development to test and demonstrate a CPS composed of virtual nodes (i.e., simulated elements) and real nodes (e.g., objects and their sensors and actuators). This middleware also enables hybrid approaches as combinations of virtual and real nodes on both the cyber (i.e., software) and physical side.

Blech et al. call the combination of existing physical elements and software simulation “cyber-virtual systems” [4]. The authors highlight the importance of “visualization, simulation and validation of cyber-physical systems in industrial automation during development, operation and maintenance” supported by Hardware-in-the-Loop (HIL) approaches. HIL is known from the domain of embedded systems where hardware “parts of a system are simulated in software to test a distinct system component.”

Kim et al. argue that conventional HIL simulations are not suitable for CPS as these simulations are usually built for specific systems in non-distributed environments [11]. Therefore the authors propose a human-interactive HIL simulation framework for CPS. It supports a fully distributed and scalable environment that connects human-interactive (i.e., physical) devices for input, distributed simulators and a physical system as the target to be tested.

3 Project Approach

The idea of this project is to interweave aspects from the above mentioned related work with an experimental setup while focussing on the primary goal to learn about the engineering process of CPS. Following this idea, we aim to develop problem solving techniques and guidance for work organization in CPS development projects. The actual results of the implementation are secondary. Therefore the following section is focussed on the work items and process and leaves out some technical details of the solution.

3.1 Project Setup

The project team consisted of software engineers, mainly the authors and one student who contributed to specific work items concerning simulation and intersection protocol algorithms. While high-level requirements were clearly set, detailed analysis of the real requirements was not possible and hence the team followed an agile approach [19, p. 57ff.] with weekly or bi-weekly meetings and an incremental development of features to explore technical boundaries (see Fig. 1). From the beginning, it was clear that due to resource constraints, the intersection needed to be modeled: no real cars, let alone a real intersection were available. Lego Mindstorms was chosen as framework for the physical model and accordingly the intersection needed to be built in a scaled-down indoor environment. Benefits and disadvantages of this approach will be discussed in the course of this chapter. The structure resembles the phases of the project and observations and lessons learned are outlined in the context of the sprint phase where they occurred.

3.2 First Sprint

The first sprint primarily served as a basis for subsequent work and guidance for the team members. In this regard, the sprint was primarily composed of two elementary tasks. First, the development hardware was examined and configured. Second, concepts and models to digitize the physical components were created.

In terms of hardware, we started with NXT and EV3 robots from the Lego Mindstorms series. The rationale behind this decision was that this is a proven framework for experiments in robotics. For better control over the hardware we used the LeJOS operating system [8], which allows execution of arbitrary Java software as opposed to the original Lego OS. After the construction of a prototypical vehicle, we started to implement the on-board software. In our concept, the vehicles should communicate their location to a central server in order to retrieve control information. The communication was to take place from the vehicle to a computer via Bluetooth. On the computer, a bridge software needed to be set up, which forwards the bluetooth communication via REST calls to a server. The NXT controllers that were used do not have the capability to implement IP-based communication themselves.

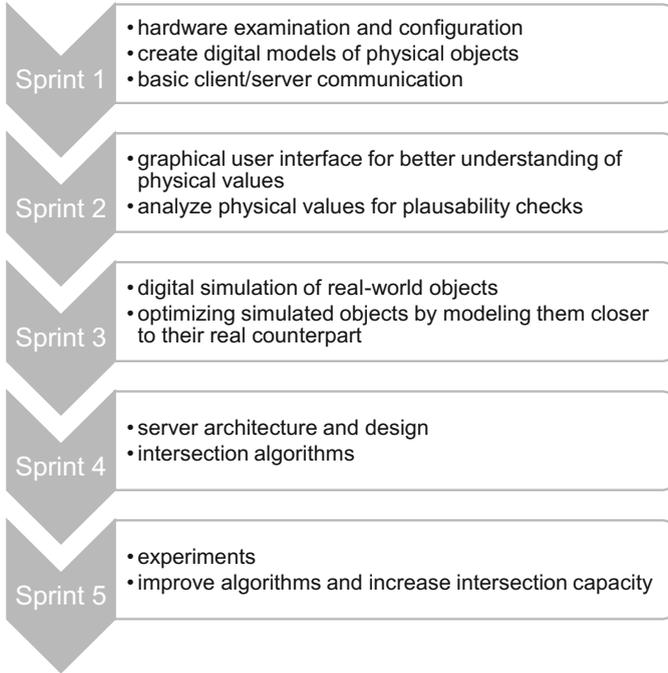


Fig. 1. Sprint overview

A special conceptual and technical challenge was the localization of the vehicles. As our experiment should be carried out indoor, GPS localization could not be used – the necessary precision of two to five centimeter for the scaled down car models cannot be achieved with normal GPS, especially not in an indoor environment. Also radio-frequency beacons are able to provide this spatial resolution. Instead, we decided to divide the lanes into fields (approximately five centimeter long), coded with different colors (see Fig. 2).

The unique color combination allowed localization of the vehicle across the lane. To read the color code, vehicles were equipped with two downward-facing color sensors whose values were sent to the server continuously. With that, the server can detect the position of the vehicle by matching its color combination and the color-coded model of the intersection.

The server component was implemented as Node.js server. In the first sprint, the server does not control any of the vehicles. Instead, only the transfer of the location from the vehicle to the server was implemented.

Lessons Learned: We learned that for the design decisions on how to abstract reality to a model, the available sensors play an important role. The characteristics and limitations of the sensors to be used need to be adequately studied to ensure that all functions can be implemented later. The sensors' specified

capabilities are therefore a valid starting point for the model. However, the specification and documentation of the sensor must be read, tested and validated to ensure they can be used in the scenario – we found our designated color-sensors to massively underperform in the target environment. Also, we had to change the physical mounting and adjust the height of the color sensors to reach a better performance. Eventually the experimentation led to a reduced set of seven identifiable colors which was fed back to the modeling task in order to color code the intersection.

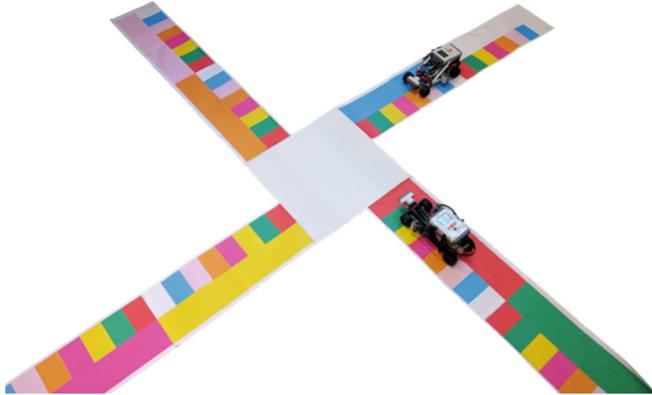


Fig. 2. Intersection prototype with robotic cars (Color figure online)

3.3 Second Sprint

In the second sprint, we focussed on creating a visualization to enable a more productive development and testing. While the messaging of the detected vehicle location to the server had already been implemented, it could only be written out to the console – a not well-readable form of information that makes the system difficult to debug. Thus, errors could not be detected directly, because they were not obvious to the human eye – especially, if the incoming values seemed plausible. To solve this problem, a graphical user interface (GUI) component was implemented. The GUI should enable developers to visualize the vehicle locations known by the server. Therefore, we constructed a digital twin of the intersection. Technologically, the implementation of the GUI was done in JavaScript to keep it close to the server codebase (especially for data structures and related code). HTML and the canvas element were chosen as native visualization means in this technology stack. As a result, movements on the physical intersection could be compared to the information of the server which were visualized on the screen. Each vehicle was represented by a simple rectangle in the digital model. The shape contains vehicle information such as location (including the color combination), speed, vehicle length, etc.

Lessons Learned: First, we depicted real-world objects in a machine readable format on the server. The problem here is that this model is not comprehensible enough for human developers in order to detect errors at first sight. Therefore we decided to visualize the created model as well. Even if this meant additional work, we invested the required time to accelerate the ongoing development process. HTML canvas proved to be a capable and well-performing, however not very comfortable option develop the GUI. Feedback from team members developing the intersection algorithms suggest that in fact identification of errors and troubleshooting got more productive and effective. Hence, we conclude that human readable models and visualizations help to structure the CPS development process and to increase its efficiency.

3.4 Third Sprint

It quickly became clear that the development was slowed down by the exclusive use of physical cars. The recurrent placement of vehicles increased the test effort further and slowed down debugging. The team therefore decided to simulate vehicles. Thus, this sprint focussed on developing a virtual car component. The idea was that the virtual instance of the vehicle would use the same interfaces as the physical robotic car. This way, the server does not know if a connected vehicle is a real or virtual (it is a blackbox). Other than the robotic vehicle, the virtual car component included the digital model of the intersection. To simulate driving behavior, it sent messages to the server with color codes from the model in the given order. Moreover, also the virtual vehicle could be controlled by commands from the server. Same as the robot, it would react to target speeds set by the server and behave accordingly. Effectively this meant that at higher speed, the color combinations were sent at shorter intervals to the server. At the same time, the experiments with robotic cars showed that the physical model needed extension. In some situations, the server assumed that cars had already cleared the intersection - but in reality they were still crossing. We speculated that the robotic cars varied their speed based on battery charge and available voltage. Thus we decided to add a color code to the lanes right after the intersection. By passing this color combination, the cars could declare themselves clear off the collision zone.

Lessons Learned: CPS focus on the interaction of physical objects, hardware and software. Real-world objects are sensed and measured and the system responds to their properties. Since the physical properties change constantly, scenarios are very difficult to reproduce. Furthermore, the continuous use of physical objects in the development process creates an enormous test and debug workload on software developers. The simulation of these objects can reduce the workload and accelerate the development process. The fact that simulated components are at work should be intransparent to other CPS components, otherwise the validity of the simulation is at risk.

3.5 Fourth Sprint

Once we were able to transmit the locations of real and virtual vehicles without errors to the server, the performance of any intersection algorithm could digitally be visualized on the screen. The next step was to deal with the control of the vehicles. In our setup, the server stored all locations of all vehicles at all times. This centralized server should now decide how fast each vehicle should move. The primary goal of any intersection control should be that there are no collisions, the secondary goal being optimal use of the road capacity. In this sprint, we implemented two competing routing algorithms that pursued different strategies in reaching the aforementioned goals.

The algorithms were designed as modules in the server, so that they could easily be switched and compared. Both routing algorithms got access to the location of all vehicles, their main task being the calculation of a target speed for each vehicle. The server then sent this information to the vehicles that adjust their speed accordingly. By using the virtual cars we were able to test the algorithms with unlimited vehicles and observe the behavior. As stated before, the main focus was to avoid collisions.

Lessons Learned: While implementing the routing algorithms, the virtual vehicles were used successfully to simulate the intersection. We learned that without simulations, no productive development is possible, as the iteration cycles between implementation and testing are very frequent and short. Often, changing a piece of code only took a few minutes to complete. Afterwards, a brief test needed to be performed in each case. Doing these tests during implementation with real vehicles is virtually impossible because this takes too much time and effort.

3.6 Fifth Sprint

In a final sprint we experimented with the different vehicles and tried to optimize the functions of the system. Focus here was to increase the capacity of the junction without causing collisions. To make the comparison more objective, a collision counter was implemented together with the option to freeze the GUI at the event of a collision. Prior to this, longterm testing was effectively useless as collisions needed to be observed by a human. The routing algorithms' code was cleaned and partially rewritten. In the experiments we also tried to combine real and virtual cars to observe if they behaved similarly and if the intersection could handle this situation.

Lessons Learned: After we had reviewed much of the implementation using virtual vehicles, we had to perform integration tests with real hardware. This showed how important it was that the virtual vehicles act as precisely as possible like their physical counterpart. The test on real hardware was completed quickly – only small changes to the code were necessary, e.g. to synchronize the speed of virtual cars to real cars. The routing algorithms and server components did not require further changes.

4 Project Results and Critical Acclaim

The project ended with a successful test of the intersection with both robotic and virtual cars, as well as a combination. At this stage the following artifacts have been developed:

- Physical model of the intersection and three physical model cars
- Bluetooth bridge for cars connected to the server module
- Two intersection protocol implementations
- Digital model of the intersection and graphical user interface
- Virtual car simulation module

With this setup, different use cases are supported. The virtual components together with the GUI proved very valuable to simulate behavior and visualize problems. In this use case, the virtual cars are controlled by the server according to the given protocol. Algorithms can be interchanged and collision numbers compared. The same applies when physical cars are used. These also are controlled through the server and the bluetooth bridge component. The latter being necessary for coupling of devices and tunneling of IP-based communication with the server component. Our experiments showed that the concept of indoor localization with color coded lanes worked rather precisely. The physical robotic cars worked well, although their physical properties in terms of speed and maneuvering capabilities are certainly in need of improvement. Although our analysis lacks detailed statistics, the results are promising as the throughput of the intersection seems ample and collisions could not be observed.

One main limitation of the scenario is that cars do not change their direction, i.e. they do not turn. While it is certainly possible to build Lego cars and software that enables precise turning maneuvers, we decided to not spend time on this feature. Secondly, we did not use ultrasonic distance control, despite the availability of the sensor in the Lego NXT framework. With this, it would have been relatively easy to ensure that a certain safety distance is kept by cars. We decided against this in order to focus on the algorithmic quality, however these sensors are widely used today for park distance control and are also part of the sensor package in autonomous vehicles. Another limitation is that we decided not to use vehicle to vehicle (V2V) communication. V2V is speculated to be a major feature of connected cars and enable better peer-to-peer coordination (amongst other benefits). This also would make the algorithmic control problem more interesting. While all these features would add realism and more complexity to examine the algorithms, it would have been just more iterations through physical and virtual models and software. Hence, we conclude these would not have added value for our focus area: the engineering process.

5 Results on Methodology and Engineering Process

Our observations are presented in two steps. First we elaborate on the differences between a CPS development process and an information system development process. Then we take those differences and generate a preliminary model for a CPS software engineering process and its phases.

5.1 Observations on CPS-Specific Tasks

CPS-specific tasks are naturally connected to the physical aspects otherwise missing in a software project.

The first step in our experiment was the construction of a viable physical model. This is due to the design decision to use building blocks instead of more integrated components. Certainly the idea to use Lego Mindstorms/NXT is a compromise between fabricating a physical model from scratch and choosing a fully integrated system like Anki Overdrive. As described, sensors and actuators need special attention. To information system developers it may come as a surprise that the sensor's capabilities, to a large extent, dictate the data model and even many functional aspects. In our case, the color sensor limitations influenced not only the physical architecture of the car but also the model of the intersection, such that localization was finally possible with the data generated. These constraints can be interpreted as technical requirements, well known to software engineers. However, their nature is different as they need to be developed and deduced from the physical processes in scope and are not given by stakeholders during the classical requirements elicitation phase methods such as interviews or scenarios [19, p. 99ff.]. This physical aspect is usually not present in the construction of information systems. It creates a greater uncertainty in the specification phase which leads to even more iterations between requirements and architecture. These iterations are similar to component tests, where a subsystem is tested individually in a prototypical and explorative manner.

Second, after the physical model has been developed and translated into an information and data model, the software implementation phase started to influence physical aspects again. When the first algorithm was tested with physical cars, we found that cars would collide because the algorithm assumed they had already cleared the intersection. So, in order to have a robust method to determine if the intersection was clear or not, it was decided to add a unique color code after the intersection to each lane. Cars would send this marker to the server and thereby declare that they leave the scope of control. Clearly, this measure could have been foreseen at an earlier stage. But to us it only became apparent when tested with physical model cars, that behaved physically correct. While it can be credited to a lack of experience in building CPS, it does seem realistic to assume that implementation aspects of CPS are likely to be overlooked until first tests of the integrated system are conducted. While it is certainly desirable to follow a holistic plan-driven approach in any safety-critical environment [19, p. 57] such as traffic control, the heterogeneity of CPS might render this ambition unfeasible. Thus, the CPS engineering process must not prohibit later adjustments to aspects of the system but rather enable change.

Third, the behavior of the physical cars was again modeled into a virtual simulation component. This component acted like the physical car and communicated with exactly the same protocol reading color codes and processing speed commands. As a result, it was not transparent for the server if a car was virtual or real. Being able to instantiate an almost arbitrary amount of (virtual) cars made testing and simulating the intersection algorithms a lot more convenient

and productive. However, we found it difficult to recreate the exact physical behavior of the cars in terms of acceleration and speed. The physical cars would sometimes come off track or show faster or slower speeds (possibly due to higher or lower battery charge states). Nonetheless, the virtual cars made it possible to develop algorithms in a more reasonable time. Simulation is therefore both an accelerator, as well as a threat to development process. The danger is mainly rooted in the fact that a simulation can never fully account for the variability of real world applications and therefore cannot guarantee faultless operation.

5.2 Preliminary Model of CPS Software Engineering Phases

Throughout the different phases of our development, we saw strong parallels to the *Twin Peaks* model [15]. *Twin Peaks* emphasizes the interrelation of requirements and architecture, the notion being that there is a permanent exchange of information between the two. Moreover, *Twin Peaks* conveys the idea that one starts with a general understanding of requirements and architecture and iteratively generates a detailed view.

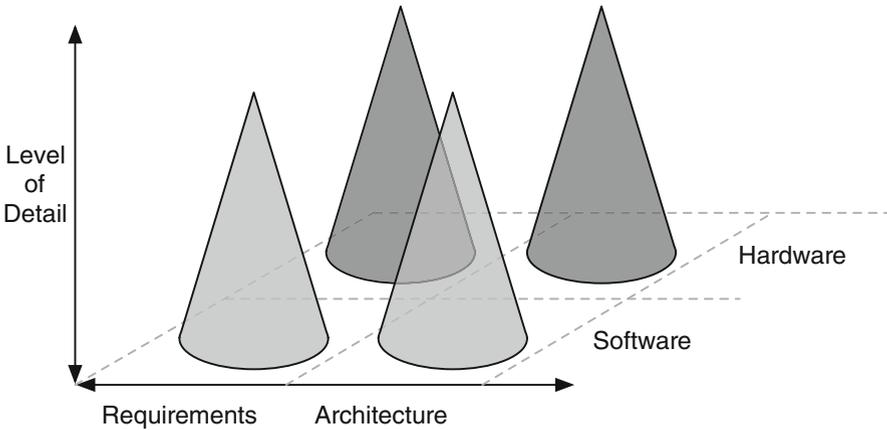


Fig. 3. *Double Twin Peaks* overview

Based on our observations, we propose to add the notion of physical (real-world) objects to this model. We found that often physical properties and constraints were only discovered when applying our architectural decisions. Therefore the iterations are not purely software-based but clearly there is a hardware-related loop to the CPS engineering process. As with *Twin Peaks*, this is a bidirectional information gain: software needs to be adapted when physical aspects change and the physical part of the CPS evolves when new software is implemented and tested. For example the requirement to localize vehicles indoors led to the development of a physical color-code schema. This in turn incurred that vehicles needed color sensors. Experiments with the latter showed that the positioning of the sensors

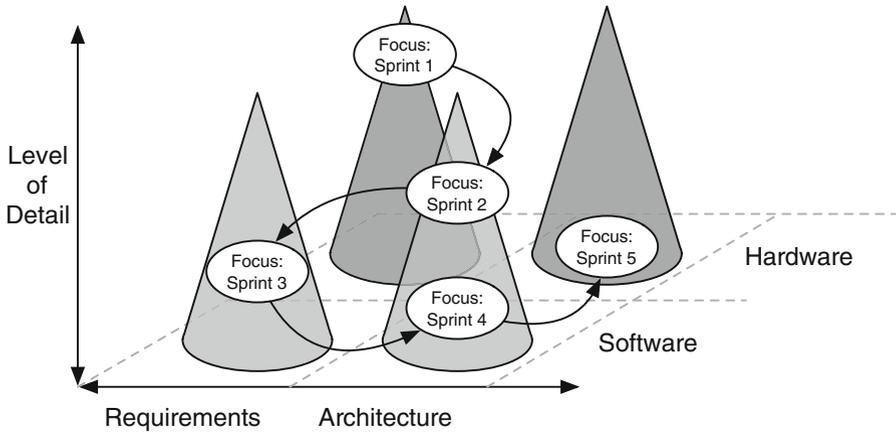


Fig. 4. Illustration of the observation that focus areas of sprints alternated over the course of engineering through different domains in *Double Twin Peaks*.

is crucial to obtain enough reliable data. The number of colors supported by the sensor influenced the software data model, and so on. We think it is crucial to understand that CPS components cannot be developed separately, but in fact the development of cyber and physical parts are closely aligned and interdependent. Therefore we present the *Double Twin Peaks* model, as shown in Fig. 3 below. In the foreground, it depicts the Twin Peaks of software requirements and architecture, adding a layer for the respective doubles in the physical domain – hardware requirements and hardware architecture.

Double Twin Peaks emphasizes the strong influence of physical properties on the design and development of CPS. As described in the original Twin Peaks publication, similar attention needs to be put on software requirements, architecture as well as hardware design and physical constraints. It can be hypothesized that as with software there exist CPS-patterns, i.e. solutions that can be applied in a range of similar problems. What can be said for sure is that a modular, well-defined and reusable system architecture is desirable for CPS in order to cope with complexity and shorten development cycles. However, this ambition is challenged as the physical part of CPS tend to be specifically tailored to the physical environment and relevant constraints. As physical components influence the software part, the entire CPS drifts towards a specific solution. Figure 4 depicts the described development journey in the *Double Twin Peaks* model. Despite the limited number of sprints, we experienced a constant interchange between domains which did not follow the textbook approach in software engineering.

While *Double Twin Peaks* is a preliminary result and needs validation, the generated insight might help developers aiming to build CPS to realize possible faux pas such as strict separation of teams or lack of communication.

6 Conclusion and Future Work

In this paper, we have described the development of a cyber-physical intersection management system. Certainly the prototype was very simplified and spared external factors such as pedestrian behavior, unexpected obstacles or human driver behavior. However, our models first priority purpose was not to serve as a realistic intersection simulation but rather generate initial insights into the CPS engineering process, which to the best of our knowledge are otherwise not available. The experience gathered led to the preliminary version of the *Double Twin Peaks* model. Here, we argue that not only software requirements and architecture influence one another but extend our view to the physical domain. Hardware and physical environment impose constraints on CPS software. But this is not a one-way road – for example, we have shown that design decisions in the software domain may lead to new requirements for hardware components or new ideas how to interact with the physical environment. We are currently developing a follow-up project. With this second generation prototype we would like to validate the *Double Twin Peaks* development model presented previously. The model should consistently be applicable throughout the phases of the CPS engineering life cycle. The focus is requirements engineering, architectural design, implementation and concurrent testing.

In this prototype setup we had used building blocks from the Lego NXT robotics framework. From a functional perspective, it is desirable to move away from this system as the NXT control units lack computing power and modern communication. In addition, to improve the cars speed and precision of movements, the mechanical design would need to be generally revised. Thus for the following project, we aim to use Anki Overdrive [9].

Higher speed and greater flexibility are not the only changes compared to NXTs – Anki vehicles are equipped with modern communication technology, optical sensors and artificial intelligence. The cars are fully-integrated devices whose software can not be customized [2]. Hence, a major benefit of using Anki is that we are able to observe if our findings still hold true for a more integrated platform, where most physical aspects are predetermined.

Thus, the focus of future research is no longer the software implementation of the vehicle itself. Instead, we will concentrate on the network of vehicles and the intelligent intersection control algorithm. With this slight change in focus, we are keen to see if the hardware layer in the *Double Twin Peaks* model is still relevant. Apart from that, Anki brings further possibilities to extent the project scope: up to four vehicles can travel side by side on the track. The track could therefore be divided into two tracks per direction, resulting in new possibilities for the control system and more realism compared to real world traffic. Moreover, we are planning turn maneuvers on the intersection to further increase the option space for the prototype.

Acknowledgments. This work has been supported by the European Community through project CPS.HUB NRW, EFRE Nr. 0-4000-17.

References

1. Al-Jaroodi, J., Mohamed, N., Jawhar, I., Lazarova-Molnar, S.: Software engineering issues for cyber-physical systems. In: 2016 IEEE International Conference on Smart Computing (SMARTCOMP), pp. 1–6, May 2016. doi:[10.1109/SMARTCOMP.2016.7501717](https://doi.org/10.1109/SMARTCOMP.2016.7501717)
2. Anki Inc.: Anki drive SDK (2016). <https://github.com/anki/drive-sdk>
3. Azimi, R., Bhatia, G., Rajkumar, R., Mudalige, P.: Intersection management using vehicular networks (2012). <http://papers.sae.org/2012-01-0292/>
4. Blech, J.O., Spichkova, M., Peake, I., Schmidt, H.: Visualization, simulation and validation for cyber-virtual systems. In: Maciaszek, L.A., Filipe, J. (eds.) ENASE 2014. CCIS, vol. 551, pp. 140–154. Springer, Cham (2015). doi:[10.1007/978-3-319-27218-4_10](https://doi.org/10.1007/978-3-319-27218-4_10)
5. Bures, T., Weyns, D., Berger, C., Biffel, S., Daun, M., Gabor, T., Garlan, D., Gerostathopoulos, I., Julien, C., Krikava, F., Mordinyi, R., Pronios, N.: Software engineering for smart cyber-physical systems - towards a research agenda: report on the first international workshop on software engineering for smart CPS. ACM SIGSOFT Softw. Eng. Notes Arch. **40**(6), 28–32 (2015). doi:[10.1145/2830719.2830736](https://doi.org/10.1145/2830719.2830736). ISSN 0163–5948
6. Derler, P., Lee, E.A., Vincentelli, A.S.: Modeling cyber-physical systems. Proc. IEEE **100**(1), 13–28 (2012). doi:[10.1109/JPROC.2011.2160929](https://doi.org/10.1109/JPROC.2011.2160929). ISSN 0018-9219, 1558-2256. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5995279>
7. Gao, P., Kaas, H.-W., Mohr, D., Wee, D.: Automotive revolution - perspective towards 2030, January 2016. <http://www.mckinsey.com/~media/mckinsey/industries/high%20tech/our%20insights/disruptive%20trends%20that%20will%20transform%20the%20auto%20industry/auto%20202030%20report%20jan%202016.ashx>
8. Griffiths, L., Shaw, A., Bagnall, B.: LeJOS, Java for lego mindstorms (2009). <http://www.lejos.org>
9. Heidloff, N.: Node.js controller and MQTT API for Anki over-drive, May 2016. <https://github.com/IBM-Bluemix/node-mqtt-for-anki-overdrive>
10. Kim, K.D., Kumar, P.R.: Cyber-physical systems: a perspective at the centennial. Proc. IEEE **100**, 1287–1308 (2012). doi:[10.1109/JPROC.2012.2189792](https://doi.org/10.1109/JPROC.2012.2189792). ISSN 0018-9219
11. Kim, M.J., Kang, S., Kim, W.T., Chun, I.G.: Human-interactive hardware-in-the-loop simulation framework for cyber-physical systems. In: Second International Conference on Informatics and Applications (ICIA), pp. 198–202 (2013). doi:[10.1109/ICoIA.2013.6650255](https://doi.org/10.1109/ICoIA.2013.6650255)
12. Lee, E.A.: CPS foundations. In: DAC 2010 47th ACM/IEEE, pp. 737–742 (2010). doi:[10.1145/1837274.1837462](https://doi.org/10.1145/1837274.1837462). ISSN 0738-100X
13. Lee, E.A.: The past, present and future of cyber-physical systems: a focus on models. Sensors **15**(3), 4837–4869 (2015). doi:[10.3390/s150304837](https://doi.org/10.3390/s150304837). ISSN 1424-8220. <http://www.mdpi.com/1424-8220/15/3/4837/>
14. Leitao, P., Colombo, A.W., Karnouskos, S.: Industrial automation based on cyber-physical systems technologies: prototype implementations and challenges. Comput. Ind. (2015). doi:[10.1016/j.compind.2015.08.004](https://doi.org/10.1016/j.compind.2015.08.004). ISSN 01663615
15. Nuseibeh, B.: Weaving together requirements and architectures. Computer **34**(3), 115–117 (2001). doi:[10.1109/2.910904](https://doi.org/10.1109/2.910904). ISSN 0018-9162. <http://dx.doi.org/10.1109/2.910904>

16. Paczesny, T., Domaszewicz, J., Konstańczuk, P., Milewski, J., Pruszkowski, A.: Between simulator and prototype: crossover architecture for testing and demonstrating cyber physical systems. In: Pentikousis, K., Aguiar, R., Sargento, S., Agüero, R. (eds.) MONAMI 2011. LNICSSITE, vol. 97, pp. 375–385. Springer, Heidelberg (2012). doi:[10.1007/978-3-642-30422-4_27](https://doi.org/10.1007/978-3-642-30422-4_27). ISBN 978-3-642-30422-4
17. Petnga, L., Austin, M.A.: Safe traffic intersections: metrics, tubes, and prototype simulation for solving the dilemma zone problem. *Int. J. Adv. Syst. Meas.* **8**, 241–254 (2015)
18. Rüchardt, D., Bräuchle, C.: A large software vendor’s view on cyber physical systems. In: 3rd International Workshop on Emerging Ideas and Trends in Engineering of Cyber-Physical Systems (EITEC), pp. 29–34 (2016). doi:[10.1109/EITEC.2016.7503693](https://doi.org/10.1109/EITEC.2016.7503693)
19. Sommerville, I.: *Software Engineering*, 9th International edn. Pearson, Boston (2011). ISBN 0137053460
20. Tachet, R., Santi, P., Sobolevsky, S., Reyes-Castro, L.I., Frazzoli, E., Helbing, D., Ratti, C.: Revisiting street intersections using slot-based systems. *PLoS One* **11**(3), e0149607 (2016). doi:[10.1371/journal.pone.0149607](https://doi.org/10.1371/journal.pone.0149607). ISSN 1932-6203. <http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0149607>
21. Wuthishuwong, C., Traechtler, A.: Vehicle to infrastructure based safe trajectory planning for autonomous intersection management. In: 13th International Conference on ITS Telecommunications (ITST), pp. 175–180 (2013). doi:[10.1109/ITST.2013.6685541](https://doi.org/10.1109/ITST.2013.6685541)



<http://www.springer.com/978-3-319-56996-3>

Ambient Intelligence

13th European Conference, Aml 2017, Malaga, Spain,

April 26-28, 2017, Proceedings

Braun, A.; Wichert, R.; Maña, A. (Eds.)

2017, X, 279 p. 115 illus., Softcover

ISBN: 978-3-319-56996-3