Cloud-Based System Architecture for Driver Assistance in Mobile Machinery

O. Koch , B. Beck , G. Heß[*], C. Richter[*], V. Waurich[*], J. Weber , C. Werner[**], and U. Aßmann[**]

Chair of fluid-mechatronic systems, Technische Universität Dresden, Dresden, Germany
E-mail: benjamin.beck@tu-dresden.de, oliver.koch@tu-dresden.de, mailbox@ifd.mw.tu-dresden.de
[*]Chair of construction machinery, Technische Universität Dresden, Dresden, Germany
E-mail: georg.hess@tu-dresden.de, christian.richter1@tu-dresden.de, volker.waurich@tu-dresden.de
[**]Chair of software technology, Technische Universität Dresden, Dresden, Germany
E-mail: uwe.assmann@tu-dresden.de, christopher.werner@tu-dresden.de

Abstract

Using the example of a wheel loader, this paper presents a cloud-based system architecture enabling intelligent machine behavior. In order to achieve the final goal of a fully automated bucket filling routine, while controlling the loaders engine, travel drive and attachment, different levels of automation are processed gradually. As a first step towards automation, driver assistance can be considered. The paper explains the design choices for a cyber-physical-system architecture in the context of construction machinery. This comprises the communication framework and the cloud-application for self-adapting systems (i.e. the MAPE-K loop). As a validation of the architecture and as a demonstrator, a driver assistance functionality has been implemented. Calculations from the cloud-application give the operator feedback about efficiency, loads and task status. A developed visualization app on a tablet serves as user-interface. Concurrent simulation allow an optimization of control algorithms for the machine control and the trajectory planning. Besides changing the parametrization of the underlying models, a solution to change ECU-code at run-time without interrupting the operation is presented. The developed system architecture is the basis for further implementations of adaptive algorithms that improve future machine operation.

Keywords: cloud computing, smart metering, IIoT, construction machinery, systems architecture

1 Introduction

Applications in the field of automation and cyber-physical systems (*CPS*) are still a recent topic of research. For mobile machinery, the development tendencies in this direction are obvious, both in industry and academia.

Besides the typical challenges of control engineering in hydraulic systems and the abstraction of operating processes for automation purposes, new challenges arise, i.e. to design a suitable system architecture for CPS applications in the domain of construction machinery.

Within this paper, a proposal for a cloud-based architecture to operate an automated wheel loader is described. The goal is to implement an assistance function to automatically fill the wheel loader's bucket and design an appropriate driver interface. As the working conditions for a wheel loader are rarely constant and predictable, a self-adaptable and experience-based automation strategy is feasible. This means, that the algorithms for detecting the pile, computing a target trajectory as well as controlling the loaders engine and hydraulic system are parametric and can vary throughout various operation cycles. Increasing flexibility and guaranteeing self-adaption in combination of optimization requires to change the machines control algorithms. An interruption of the machine usage for changing a control strategy will not be accepted by the operator. Therefore, a run-time exchange of machine software is necessary. To collect, store and process the required data, a cloud-based network architecture has been implemented.

The availability of a comprehensive network that is able to monitor the machinery and to manipulate its operating logic, reveals new fields of applications. As a typical practice, additional information of the machine and its working conditions can be provided for the driver. The processing of raw data to meaningful information (i.e. called smart metering) integrates well with the proposed architecture and creates additional value for the driver as well as for the monitoring site operator. These assistance features are the first steps to approach the goal of an automated machine and it serves as a proof-of-concept for the implemented network architecture.

The following chapters will describe the implemented system architecture in detail. Chapter 2 describes the initial situation and the state-of-the-art in the field of CPS in construction machinery. Chapter 3 gives an overview of the applied architec-

ture and chapter 5 deals with the software adaption strategy on run-time. In chapter 4, the driver assistance functionality is described and finally, chapter 7 summarizes the paper and gives an outlook.

2 State of the Art

2.1 Levels of Automation

Designing a completely autonomous construction machine from scratch demands enormous efforts. In order to approach this goal gradually, a development starting from a non-automated system seems much more realizable. The levels of automation of decision and action selection by [1] can serve as a guidance on the way to an automated construction machine. A coarse classification, based on the SAE-standard J3016 [2] transferred to mobile machinery depicts as follows:

- **No Automation**
  The driver monitors and controls the system at all times and is fully responsible to move into a safe state.
- **Driver Assistance**
  The system partly sets the controls whereas the driver has to monitor the system at all times and is able to control it manually.
- **Partial Automation**
  The system sets the controls partly over a certain period of time whereas the driver has to perform the remaining tasks at all time.
- **Conditional Automation**
  The system sets the controls completely over a certain period of time whereas the driver has to monitor the system and needs to be ready for control at all times.
- **Highly automated**
  The system sets the controls completely over a certain period of time whereas the driver has to take the control after automated operation.
- **Fully automated**
  The system sets the controls completely whereas the driver does not need to monitor the system. The driver can take control but the system is able to go into a safe state at any time.

In industrial hydraulic applications, the boundary conditions like temperatures, process forces and duty cycles are mostly good known. Furthermore, there is a less cost pressure because investments into plants recoup early due to large quantities of produced products. This results in a higher level of automation like shown on an example of a hydraulic power unit in [3]. But the task of monitoring and analyzing data is application independent. Concerning mobile hydraulic applications, just a few assistance functions like a return-to-dig routine in a wheel loader are integrated in mobile machines (Driver Assistance), nowadays. Most of the tasks are operator controlled (No Automation). This is mainly due to the diverse working tasks, the harsh environmental conditions like vibrations, dust and a wide temperature range in combination with a required high availability of the machine. However, the continously increasing demands on productivity and efficiency require the steps of automation, because the operator cannot have an overall comprehension of the process and additionally has a decreasing productivity due to an increasing tiredness. An example is the filling and dumping cycle, which can be repeated sufficiently often in order to apply automation in a productive way. As stated in [4], the efficiency of wheel loader bucket filling strongly depends on the trajectory of the bucket. Inexperienced drivers are likely to move too much soil through the pile and hence, the filling is expensive in terms of energy. An energy-efficient trajectory would be to slice a thin chip of soil until the bucket is filled [5]. Therefore, the bucket filling is a worthwhile task for automation.The key challenges in automation and tele-remote operation of earth-moving machines concerning these levels of automation are summarized in [6]. Especially, the short-loading cycle of a wheel loader with an in-depth review of different automatic bucket loading strategies is analyzed. The authors also address the problem of material description as well as communication aspects. As a result, they suggests semi-automation as short to mid-term solution for earth moving machines. For integrating the levels of automation into mobile machinery, the following tasks have to be solved:

- Path planning for driving
- Collision detection, avoidance and navigation
- Path planning for the work equipment

Thereby the following requirements have to be taken into account [6]

- Performance of the work task (fill factor and cycle time)
- Fuel efficiency
- Safety

According to [6] there are many publications on modeling for control, automatic loading, pile characterization, localization and navigation as well as path planning. Volvo has researched on autonomous machines for more than one decade. In [7] they demonstrated how an autonomous wheel loader together with an autonomous articulated hauler reached 70% of a skilled operator's performance during a predefined cycle in an asphalt mine. The autonomous control algorithm and the autonomous bucket control are described by [8, 9]. However, the paths are predefined, there is no communication between the machines and no process optimization. Furthermore, an industrial PC along with a Simulink-based control algorithm is used [8], which are suitable for prototyping but not for commercial applications. This example shows possibilities in the development of mobile machines. Additional, autonomous functionalities or services offer novel innovations. The variety of the applications is almost unlimited. Today, it is easily possible to adapt a parameter in a control algorithm without interrupting the machine. It guarantees a little flexibility. Unfortunately, it is nearly impossible to foresee all eventualities within the design of the control algorithms. There are research approaches dealing with machine learning to consider the uncertainties. It needs a lot of computer resources that are not available on a mobile machine. The communic-

ation technologies offer a solution. Implementing a wireless network structure and bundling powerful computer resources in a cloud architecture that is able to change algorithms on a mobile machine enables a flexible and adaptable machine control structure. Learning algorithms, optimizations or condition monitoring can be executed within that cloud. The machine itself performs the specific working tasks. If a new task requires a completely new functionality that the machine is currently not able to perform, the cloud-based supervisor will learn and modify the machine's algorithms. This concept leads to the question of changing software algorithms on runtime that are currently not available on the embedded device. At the time of publication, there are no contributions to cloud-based hardware and software architectures for realizing adaptive control algorithms for mobile machines.

2.2 Cloud Solutions

Cloud platforms play a big role in the IoT world. They offer many advantages to create large IoT applications e.g. predefined analyzing functions, on demand extendable data storage, and a variable amount of computational power. Important cloud providers in this area are Amazon's AWS IoT-Platform, the Google Cloud Platform, Microsoft's Azure IoT Suite, Bosch IoT, ThingWorx IoT Platform, and IBM's Watson IoT, which are presented in [10]. All of them present different implemented modules, which internally fit perfect together to a larger IoT solution for a variety of applications. The modules offer data storage, analyzing features, and visualization interfaces. In the year 2005, IBM published a paper called "An architectural blueprint for autonomic computing" [11]. There, they present the control loop called MAPE-k. Figure 1 shows the loop in the proposed system architecture consisting of the four MAPE phases and the k representing the data storage as knowledge base. In the monitoring step all incoming data is collected and saved. The analyze phase offers functionality to correlate the data and to create high level information. Then, in the planning step, the information from the analyze step is used to compare the incoming data with the predefined objectives that should be reached and to adapt the system if necessary. The last part comprises the execution phase, which distributes new information to connected applications. This loop is integrated in most processes to create adaptable software.

IBM named its cloud architecture "open cloud architecture" [12] to create interfaces for open source standards, projects, and applications. This generates an mechanism to easily extend applications and connect them to related open source frameworks. They also implemented e.g. OASIS, W3C, IETF, and OMG standards and thereby have integrated the MQTT-Protocol. The MQTT-Protocol [13] offers existing open implementations for a lot of programming languages like C++, JAVA, or Python. It provides a publish/subscribe service with different quality metrics. On the lower level the MQTT-Protocol implements sockets that enables a fast connection and the possibility to use different data formats.

3 Overall System Architecture

Figure 1 illustrates the basic concept of the chosen system architecture consisting of the main modules wheel loader, communication interface, cloud application and user interface (i.e. a tablet).

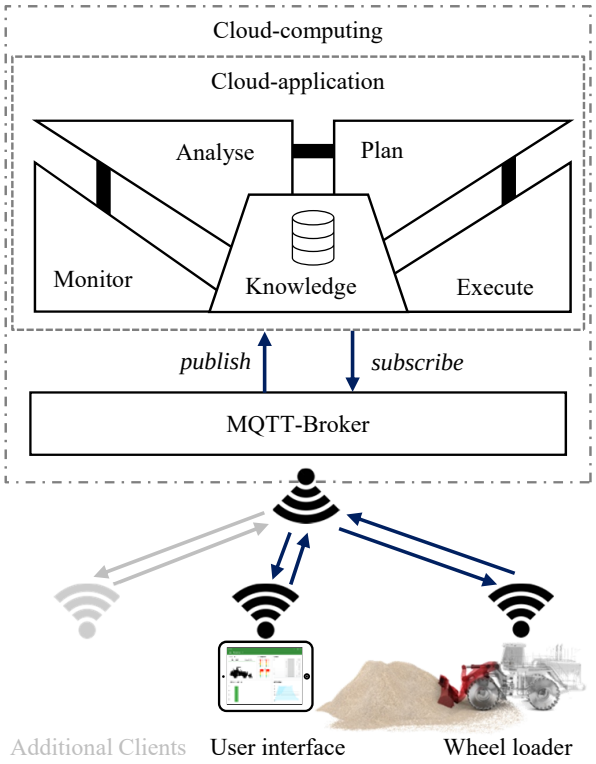

Figure 1: Overall System Architecture

3.1 Cloud Structure and Applications

The cloud is the central node which collects data from connected clients and performs the overall optimization and analysis. A client represents a specific device, a machine or a software component within the cloud which sends and/or receives data to/from another client. The MQTT-Broker plays a central role and manages all inter-communication processes between different clients. The broker itself takes only an intermediary role and routes all data messages.

All cloud-applications implementing the concept of the MAPE-k loop [14] in different abstraction levels covering the four phases: monitoring, analyzing, planning, and execution, which is depicted in fig. 1 and introduced in sec. 2.2. The data-flow between these phases is visualized by a bold line. In the *monitoring phase*, the cloud application collects all incoming sensor and machine data from the wheel loader over the communication interface, converts it into a object oriented data format and sends it to depending analyze algorithms. Furthermore, this phase saves all information into the connected database. This first implementation uses SQLite. All processes own read and write access to the knowledge database. With arrival of a certain devices first message the application creates a new database table. All following

messages will be written into this table. Subsequently, the *analyzing phase* creates high-level information by connecting recent data with historical data. In this first implementation step, the analyzing phase only computes the Smart Metering data for the wheel loader application. Additionally, the execution of Modelica-based simulation models are integrated to test this opportunity as well. They will be used for later optimization and the basis for generating changed algorithms for the wheel loader's control unit. The *planning phase* compares actual data with the target data and decides whether an algorithm has to be modified. Basically, two different options are possible. While modifying a control model's parameterization does not necessarily afford an software exchange, a replacement of a complete strategy requires downloading another software version. The last step in the cloud covers the *execution phase*. Modifications will be propagated to clients. Therefore, an FTP connection transfers the generated code to the machine and for parameterization the MQTT infrastructure is used directly.

Currently, the cloud executes every service for the implemented smart meetering application in a separate thread. It enables an independent development and test of all routines. Furthermore, all services inherit an application prototype. This already realizes basic functions to gather data from and commit data to the knowledge base. Python 3.5 serves as programming language. To reduce the amount of communication overhead and redundant software parts, merging all services in just one major cloud application is recommended and will be realized in further developments. A suitable plugin concept then allows an easy addition of new programmed services.

3.2 Network Communication

The connection between the cloud and the machine is managed by a communication interface based on the MQTT-Protocol [15]. It guarantees the creation of an adaptable, secure and scalable communication structure. Clients have to authenticate themselves to the server. The messages are encrypted by SSL/TLS. In order to add a new functionalities without changing the clouds overall implementation, the connections to services can be accomplished on the fly (Figure 1). An MQTT-Broker with a publish/subscribe interface is responsible for the clients communication. Different clients can connect themselves to the broker an subscribe and publish messages for related topics. It only requires implementing the MQTT-Client interface. For the prototypical implementation, the open source MQTT-Broker Mosquitto [16] is used.

According to individual machines, an initialization message is required to register a specific machine client. Within this initial handshake, the message protocols are exchanged. It considers varying communication processes and guarantees a certain flexibility. Reconfiguring the protocol only requires an update through a new configuration message. JSON serves as exchange format and contains application-depended key-value pairs.

3.3 Wheel Loader as a Network Client

Figure 2 illustrates the machine's control architecture consisting of two levels. The Machine Level represents the basic wheel loader functionality as it could be developed for ordinary operating tasks. The Adaptive Control Level implements additional control functionality that is used for automation and optimization during an operation.

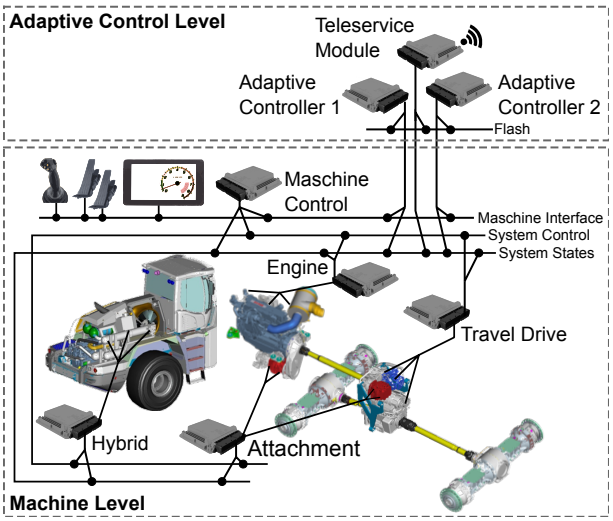

Figure 2: Wheel loader's system architecture

Therefore, the Machine Level represents a master-slave control system with three different CAN-networks. Operator devices communicate over the Machine-Interface bus with the Machine Control. This central unit interprets the operator inputs and generates the subsystems' set values according to the implemented operating strategy. Additionally, it handles functions such as control of peripheral systems, monitoring and calibration routines. Each subsystem comprises a separate controller for realizing the machine controller's demands from the System-Control bus by controlling the subsystem's integrated actors. Furthermore, subsystem-specific data acquisition and monitoring functions are implemented. They are transfered via the System-States bus.

Today, a typical mobile control unit does not usually allow changing software parts of a control algorithm on run-time. It almost always needs a reboot to start the internal flash-loader. This loader program expects getting a new complete control software from an interface and stores it in its designated memory. From there, the program will be loaded into the RAM and executed after starting the device. Operating systems usually offer a mechanism for an easy program exchange without the need for rebooting the device. Unfortunately, they cannot be installed on a today's mobile controlling unit. But novel developments usually designed for multimedia applications generate powerful embedded devices which allow running an operating system that provides this needed functionality.

This leads to two major opportunities depending on the selected embedded control unit. Using a typical mobile control device requires a control architecture consisting of at least two

redundant devices. While one controller executes the current algorithm the other unit is ready for rebooting and flashing a new program. A handshaking protocol handles switching the control units' responsibilities that the devices will change their operating state. After a software update, the recent updated control unit executes its algorithm while the other goes of and waits for a flash demand. If an embedded device with operating system shall be installed, only one device is necessary. Then, the handshake process happens between two programs on the unit itself.

The implemented Adaptive-Control-Level architecture consists of a teleservice device and two additional adaptable modules. The teleservice module acts as interface to the cloud. It handles flashing and monitoring the adaptive control units as well as gathering cloud-relevant machine data. The adaptive controllers are responsible for automated tasks that shall be modified during machine operation. They are connected to the machine interface to send control values to the master. These modules are typical mobile control units with an additional Linux board installed. It enables using both software change strategies and therefore a high flexibility. A separate installed CAN-bus (flash) handles the communication between teleservice module and adaptive control units. It shall guarantee a certain independence that an error within the adaptive control level does not directly affect the main machine behavior. All three modules listen to the System-State bus to get necessary machine information.

4 Analysis Modules in the Cloud Application

Besides several other cloud applications like on-board diagnostic (OBD), smart-metering, slip and fatigue computation will be described in the following chapters.

4.1 Smart Metering

In the context of industrial applications like mobile machinery, smart metering describes the intelligent evaluation of machine data provided by the installed sensors in combination with an appropriate visualization for the operator so that he can improve his control. Thereby the following questions have to be answered:

1. Which information or which value would improve the operator's control?
2. How can an appropriate visualization look like?
3. Which algorithms are needed to provide this data from the installed sensors?

Ideally, the algorithms should come along with the already installed sensors which are needed from the functional view. The wheel loader demonstrator already contains a variety of sensors to measure, e.g.:

- The angles in the joints of the work equipment,
- The swash plate angle of the pumps,
- The cylinder pressures and
- The shaft torque and rotational speeds at the gear output.

This application can be seen as an outlook for future development opportunities. Although this demonstrator is just a prototype and does not represent a today's commercial vehicle, the trend of installing electronic and software into mobile machines also expects an increase of sensors to detect various system states. According to the provided data for the operator, the vehicle's tire slip and a wear computation will be exemplary described in this paper.

4.2 Slip Computation

The illustration of the tire slip value λ can be divided into an optimal slip section and a section with an expected increase of tire wear. The operator shall recognize an unnecessary high tire wear and be able to reduce the wheel loader's output torque via its drive pedal. The optimum varies between 5 - 20 % depending on the environmental conditions like weather or ground conditions. This is expressed by the tire slip - friction coefficient curve displayed in fig. 3.

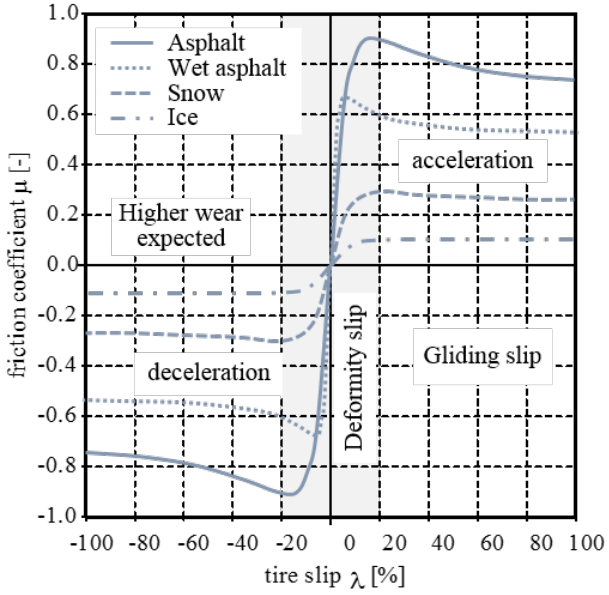

Figure 3: λ-μ-curves for different surfaces according to [17]

Because the machine only provides the output torque and rotational speed of the travel drive, there are two tasks for estimating the actual tire slip. First, an calculation algorithm is required and secondly, the machine's slip-free velocity $v_{machine}$ has to be measured. The calculation of the longitudinal tire slip λ_x is given by eq. (1), whereas n_{wheel} and D_{wheel} are the rotational speed and diameter of the wheel.

$$\lambda_x = \frac{\pi \cdot n_{wheel} \cdot D_{wheel} - v_{machine}}{\max(\pi \cdot n_{wheel} \cdot D_{wheel}, v_{machine})} \quad (1)$$

In the case of deceleration the numerator causes a negative velocity difference and the maximum function in the denominator becomes $v_{machine}$ resulting in a negative slip. On the other hand, an accelerating wheel has a positive numerator. The denominator becomes $\pi \cdot n_{wheel} \cdot D_{wheel}$ and a positive slip is returned. Furthermore, the ratios of the differential gear and the wheel hubs as well as the dynamic wheel diameter must be

taken into account. Because these values are specific machine parameters, they have to be transferred from a machine client to the cloud application during the subscription process. The tire slip calculation is implemented in the programming language Python. It gathers the gear ratios, the dynamic wheel diameter, the output torque and rotational speed of the travel drive, the actual time as well as the machine velocity from the knowledge data base. In return, the routine publishes its actual tire slip estimation as a result to the data base. Due to the wheel loader's permanently coupled wheels, the algorithm only generates one common slip value.

4.3 Fatigue and Wear Computation

With information about the current tire slip, it is possible to estimate the tire wear. A simplified first implementation in Python calculates the cumulative frequency W_{tire} if the tire slip exceeds the deformity slip range. There, due to the gliding slip, a higher wear is expected and can be estimated according to eq. (2).

$$W_{tire} = \frac{1}{T_{life}} \sum_{i=1}^{j} t_i, \text{ if } \lambda_x \geq \lambda_{limit} \tag{2}$$

Another exemplary feature is the monitoring of mechanical damage based on the calculated stress and providing it to the operator. To estimate the load acting on the structure, it is necessary to solve the equations of motion as well as the kinematics of the equipment. To reduce calculation time, the nonlinear equation system of the kinematics has to be avoided. Therefore, the angles are calculated with causal equations instead of nonlinear systems of equations. With those calculated values and the sensor data, the equation of motion of all bodies becomes a linear problem with the unknown joint forces. This system can be solved with a QR decomposition algorithm. The execution of this calculations is done in real time and thus the results can be saved in addition to the raw data on the cloud storage. To predict a failure of a component, the joint forces have to be transformed to a load spectrum. Afterwards, the fatigue can be derived. The total fatigue accumulation serves as an indicator for failure or wear.

4.4 Implementation

The algorithms have been virtually tested in a simulation environment. The next step includes the validation of the services by means of the real machine. Therefore, a sensor to measure the slip-free machine velocity must be installed. The most important use case for optimizing the tire slip of a wheel loader is presumable filling the bucket while driving into a pile. This requires a good accuracy at lower velocities. Because GPS-sensors do not fulfill this requirement [18], alternative techniques have to be discussed. Optical sensors address this problem but are cost-intensive at the same time [19, 20]. Inertia measurement units are one promising technology for mobile machines as shown in [21]. Additionally, they will be needed to compensate dynamic effects in the calculation of digging forces or loaded mass from the cylinder pressure signals. On the basis of the slip calculation, traction control algorithms can be developed in future work to reduce the tire wear and to increase the driving behavior during an autonomous loading cycle.

5 Planing Modules in the Cloud Application

5.1 Software Adaption on Run-Time

Figure 4 illustrates a basic software architecture that is applicable for both hardware variants (board with Linux operation system and mobile controlling unit). It represents a state machine and an algorithm component. This approach allows a separation of static code from variable software algorithms. While the state machine implements algorithm-independent software parts that usually do not vary between software updates, the algorithm component realizes a specific changeable strategy. The two interfaces StmInterface and Algorithm allow an interaction between them. A concrete algorithm has to provide the interface Algorithm for updating and reseting its output values. The architecture considers automated functions that prohibit an interruption during the process as well as interruptible semi-automated functions. Therefore, a concrete algorithm uses the StmInterface to lock and unlock a deactivation or a shutdown process. The teleservice module is able to send different commands to the adaptive control devices. They will be redirected to the state machine. If an activate command is received the state machine will execute the algorithm logic by calling its update function. The deactivation command disables updating the algorithm. A reboot command shuts down the controller. Restarting the device allows flashing a new control software. A change request turns the devices' activation. The currently operating algorithm will be disabled while the inactive algorithm will be enabled. Additionally, a kill command forces a reboot of the device.

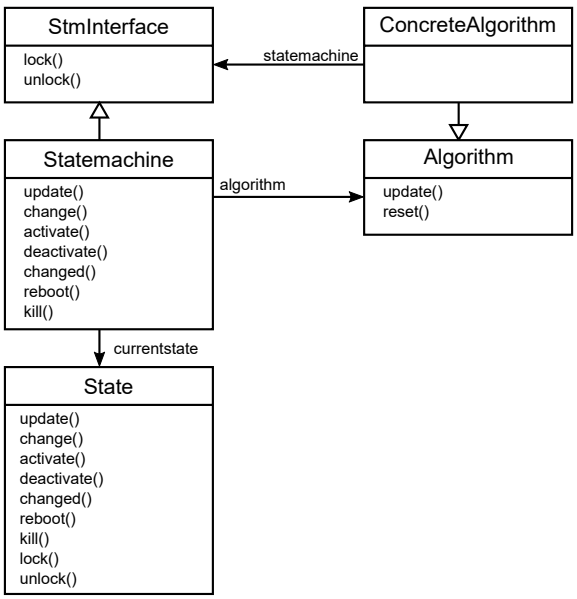

Figure 4: Components

Figure 5 defines the state machine's states and transitions. Depending on the current state, a received command causes

a different change in state. There are eight states available. After booting, the device stays inactive and the algorithm is reseted. An activation command instantly activates the installed algorithm by switching the controllers state to process whereas a change request leads to the intermediate state change requested. This synchronizing state guarantees that the currently operating algorithm turns inactive before another strategy can be performed. The new algorithm stays inactive until the device receives the notification changed from its counterpart. The process state cyclically calls the algorithms update operation. If an automated service is activated that shall not be interrupted, the algorithm can lock the state machine and the current state switches to operate. From there, various intermediate states are available. All also trigger the current algorithm to update its values until they will be unlocked. Depending on the previous command, the state machine then changes over to its desired state.

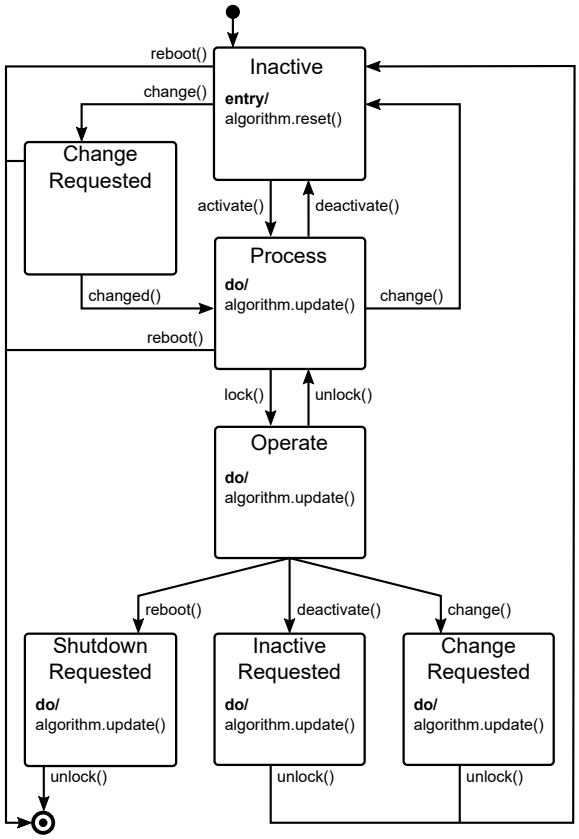

Figure 5: State machine

5.2 Flash Function

The flash function is the key point to change the run-time code on control units. It is implemented on the Teleservice device as an event based algorithm. It gets a new binary code file from the cloud via ftp connection. A separate start command initiates the flash sequence to download a new software version to a machine's control unit. It is shown in Algorithm 1. At the beginning, the routine searches for the current inactive slave module. If an inactive node is detected, the algorithm will reboot and flash it. Otherwise, a failure will be reported to the cloud. After the node successfully changed its software, restarted, and reported back its status, both control nodes receive a command to change their current operating state. From this point, the flash sequence waits for the active control device to deactivate itself. Receiving a new binary file from the cloud, again restarts the process.

Data: file = new binary flash file
Result: algorithm successful
inactive node = get inactive node;
if *inactive node is null* **then**
  start error correction;
  return *false*;
end
reboot (inactive node);
flash (inactive node with file);
while *inactive node is not restarted* **do**
  wait;
end
send change activity command;
while *active node is changed to inactive* **do**
  wait;
end
reboot (second node);
flash (second node with file);
while *second node is not restarted* **do**
  wait;
end
return *true*;

Algorithm 1: Flash Algorithm

The flash memory's write cycles are usually limited. It is useful to reduce the need for flashing a controller. Two additional opportunities are implemented. If a modification only consists of a changed parameter set, the Teleservice device is able to pass a new set from the cloud to a control node without changing the whole software code by the flash loader. Therefore, the JSON-based messages are transformed into CAN-based messages and sent to the specific node. Another possibility exists if more than one algorithm is implemented into the current control software. The abstraction to a general algorithm interface allows an easy exchange of the current logic by redirecting the state machine's algorithm reference to a new designated strategy.

6 Virtual Validation

6.1 Simulator

During the development process, it is necessary to test new functions and services quickly and with little effort. Attempts on a real machine are not always the easiest way to accomplish that. This has various reasons, e.g. a real machine is not always available, ready for use or an experiment would be to expensive. An alternative to experimental validation is virtual validation, in which the real environment is replaced by a virtual mock-up. For this purpose, a simulation tool has

been developed, which allows coupled simulations between machine and process models. The structure of the software tool is shown in fig. 6

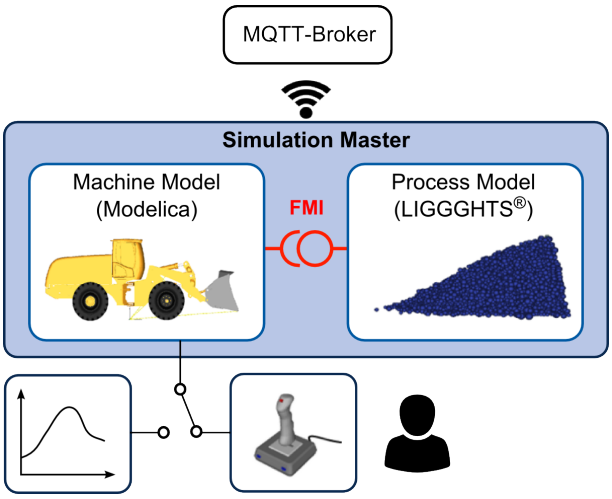

Figure 6: Interactive Machine and Process Simulator

The machine (wheel loader) was created as a comprehensive system model in the description language Modelica [22]. The model exists in different levels of detail, starting with a simple mechanical multibody system up to complex variants representing also hydraulic parts and control systems. The simulation model is exported as an *Functional Mock-up unit (FMU)* [23]. *FMU* is a tool-independent standard for the exchange of system models among both proprietary and non-proprietary software tools.

The process model, which is designed to simulate the behavior of the granular material, is based on the *Discrete Element Method (DEM)* first mentioned by P. A. Cundall [24]. For computation, the open source software *LIGGGHTS®* [25] has been applied. Due to the efficient use of multi-processor architectures, as well as the large number of available contact models, *LIGGGHTS®* is considered to be particularly powerful. In recent years, various methods to link and communicate among *LIGGGHTS®* and Modelica Models [26] or *FMUs* [27] have been developed. In order to correctly link the machine and process model, the *FMU* require defined inputs and outputs. Outputs are position, orientation and speed of the wheel loader's bucket. These values and necessary geometric information are transmitted to *LIGGGHTS®*, whereupon the resulting normal and tangential forces on the bucket are calculated. The outcome is returned to the *FMU* via corresponding input signals.

An extra simulation MQTT-client is responsible for the simulation control. It guarantees the proper data exchange after each calculation step and saves relevant results for post-processing. Additionally, the simulation master interacts with the cloud's MQTT-broker and regularly transmits current state values.

There are two possibilities for controlling the wheel loader during simulation run-time. The first option is the application of predefined trajectories. They can be calculated by algorithms or originate from real measurements and stimulate the machine model. The second option is an interactive user control. Therefore, the *Modelica_DeviceDrivers* library [28] is used. It contains models and functions for reading signals from external control devices such as joysticks or keyboards.

6.2 Smart Metering Application

An Android application running on a separate tablet gives the operator feedback about machine status, current working tasks and performance values. The device is not directly connected to the machine and needs an internet connection to show the cloud services' evaluations. Therefore, the tablet application also inherits the MQTT-Client interface and subscribes to its needed topics. The visual design consists of different modularized views. Incoming messages triggers updating all of them. Figure 7 represents an exemplary view. This consists of five modules. The illustration (1) above left shows the actual bucket's position and reports its height and the loaded mass as a numerical value. In the middle of the upper line, the bar graphs (2) informs about the estimated tire slip. Depending on the current values, the graphs change their color. While operating in a green range means an optimal traction, a yellow to red colored bar expresses a to high tire wear. The list (3) above right tells the operator about its remaining tasks. They are prioritized, named and supplemented with deadlines and objectives. The current progress status is presented as well. On the left side of the lower line, the performance graph (4) compares the efficiency of the last loading cycles. Each value represent the amount of moved material related to a cycle's energy consumption according to eq. (3). Thereby, the working efficiency P_{work} is defined as the weight of loaded material $m_{material}$ devided by the total amount of consumed energy within one working cycle E_{total} (loading a bucket starting at i.e. floating position ending in with a defined bucket height for transportation) respective to the cycle time span t_{cycle}.

$$P_{work} = \frac{m_{material}}{E_{total} \cdot t_{cycle}} \tag{3}$$

The last figure on the lower right illustrates the pile's virtual cross section (5) and its planed optimal bucket trajectory.

7 Summary and Outlook

The presented paper proposes an overall system architecture for cyber-physical systems in the context of construction machinery. The IIoT messaging protocol MQTT is well suited for these kinds of applications. Multiple clients can connect to the cloud and are integrated in the cloud application services. The cloud implementation follows the design of a MAPE-K loop. The central knowledge base stores the data history of the system. The analyse modules perform computations on raw data and the plan modules modify the logic for client. This structure has been implemented in a prototype and has been tested using a driver assistance user-interface. Physical-based simulations emulates the machine's behavior and feed the current cloud application. A monitoring visualization app on a tablet represents a service application inside

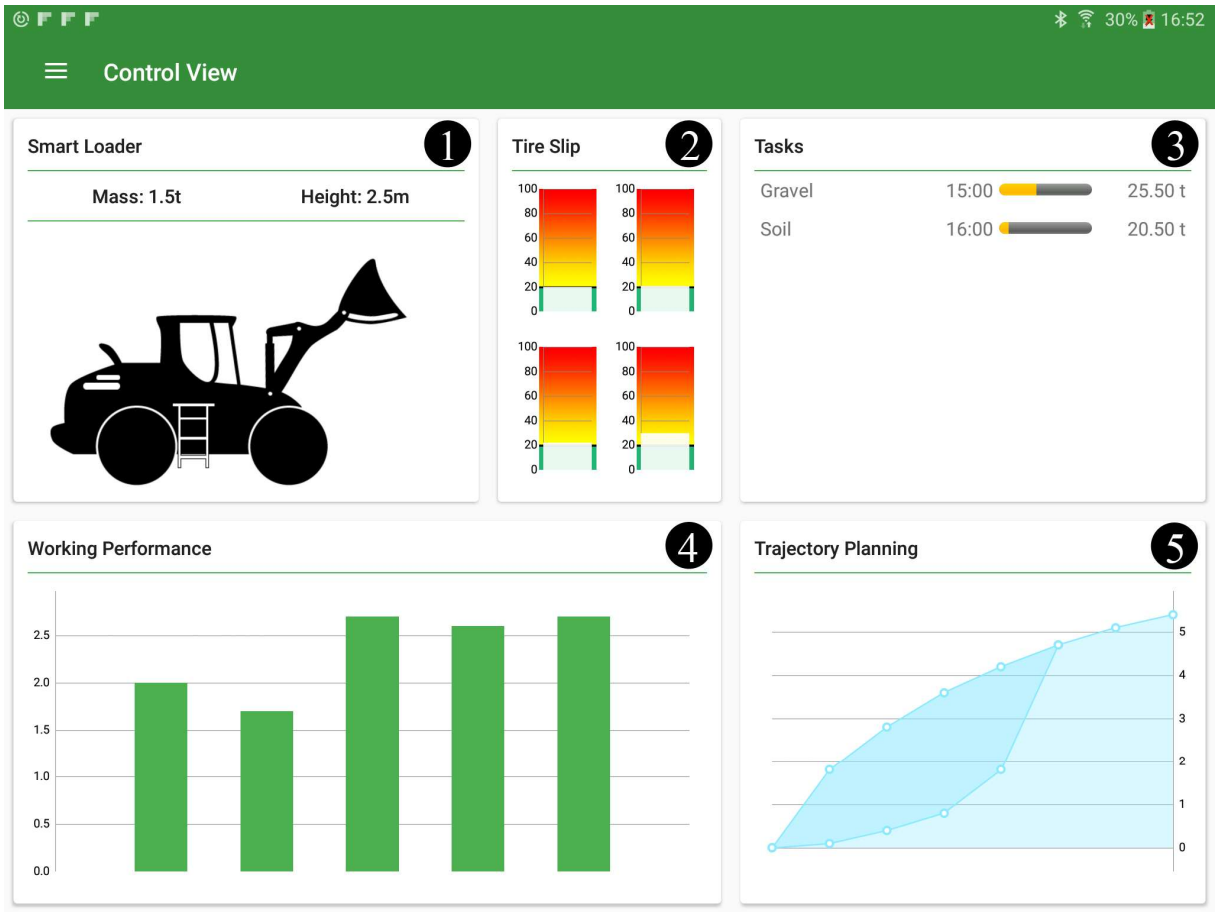

Figure 7: Application Screenshot

the CPS. The implemented cloud architecture enabled an adaptable driver assistance and defines the basis for further developments achieving high level automation functionality in mobile applications. Therefore, additional sensor technologies for environment detection are needed. With the help of the simulation models, bucket filling strategies can be easily tested but they have to be calibrated with real measurements. For further developments, an assisted, partially automated bucket filling process is obvious. In the long term, a highly automated bucket filling procedure is desirable whereas the operation of a fully autonomous wheel loader for a construction purpose seems less conceivable. The currently developed cloud implementation will be extended to an adaptable fog architecture. This means the use off more locally available computational infrastructure to split time-critical algorithms from the machine to near computational elements and not all to the cloud. This decreases latency.

Nomenclature

Designation	Denotation	Unit
λ	tire slip	%
n_{wheel}	wheel's rotational speed	min^{-1}
μ	friction coefficient	-
D_{wheel}	wheel diameter	m
$v_{machine}$	slip-free machine velocity	m/s
W_{tire}	tire wear	%
T_{Life}	life time	h
t_i	time variable	h
P_{work}	working efficiency	kg/(l·s)
$m_{material}$	weight of loaded material	kg
E_{total}	energy consumption	J
t_{cycle}	loading cycle time	s

References

[1] R. Parasuraman, T. B. Sheridan, and C. D. Wickens. A model for types and levels of human interaction with automation. *Trans. Sys. Man Cyber. Part A*, 30(3):286–297, May 2000.

[2] Sae j3016 - taxonomy and definitions for terms related to on-road motor vehicle automated driving systems.

[3] Martin Laube and Steffen Haack. Condition monitoring for hydraulic power units–user-oriented entry in industry 4.0. In *10th International Fluid Power Conference (10. IFK) March 8 - 10, 2016 in Dresden*, volume 2, pages 393–402. Dresdner Verein zur Förderung der Fluidtechnik e.V.

[4] Reno Filla. Evaluating the efficiency of wheel loader bucket designs and bucket filling strategies with non-coupled dem simulations and simple performance indicators. pages 274–292, Dresden.

[5] Reno Filla. A study to compare trajectory generation algorithms for automatic bucket filling in wheel loaders.

[6] S. Dadhich, U. Bodin, and U. Andersson. Key challenges in automation of earth-moving machines. 68:212–222, 2016.

[7] Elisabet Altin and Brian O'Sullivan. Volvo construction equipment reveals prototype autonomous machines, 2016.

[8] Jonatan Björkman. Control of an autonomous wheel loader.

[9] Anders Bergdahl. Autonomous bucket emptying on hauler, 2011.

[10] H. Derhamy, J. Eliasson, J. Delsing, and P. Priller. A survey of commercial frameworks for the internet of things. In *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–8, Sept 2015.

[11] Autonomic Computing et al. An architectural blueprint for autonomic computing. *IBM White Paper*, 31, 2006.

[12] Angel Diaz and Chris Ferris. Ibm's open cloud architecture. *IBM Corp., Armonk, New York*, 2013.

[13] OASIS Standard. Mqtt version 3.1.1, 2014.

[14] Yuriy Brun, Giovanna Di Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, Mauro Pezzè, and Mary Shaw. *Engineering Self-Adaptive Systems through Feedback Loops*, pages 48–70. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[15] Vasileios Karagiannis, Periklis Chatzimisios, Francisco Vazquez-Gallego, and Jesus Alonso-Zarate. A survey on application layer protocols for the internet of things. *Transaction on IoT and Cloud Computing*, 3(1):11–17, 2015.

[16] Roger Light. Eclipse mosquitto, 2010. An Open Source MQTT v3.1/v3.1.1 Broker.

[17] Fredrik Gustafsson. Slip-based tire-road friction estimation. 33(6):1087–1099, 1997.

[18] Heinrich Schneider and Peter Reitz. GPS zur geschwindigkeitsmessung. 51(5):264–265, 1996.

[19] LUXACT - optical sensor for non-contact displacement and speed measurement, 2013.

[20] Correvit s-motion - berührungslose optische sensoren, 2016.

[21] Chris C. Ward and Karl Iagnemma. A dynamic-model-based wheel slip detector for mobile robots on outdoor terrain. 24(4):821–831, 2008.

[22] Modelica® - a unified object-oriented language for systems modeling language specification version 3.3, 2012.

[23] Torsten Blochwitz, Martin Otter, Johan Akesson, Martin Arnold, Christoph Clauss, Hilding Elmqvist, Markus Friedrich, Andreas Junghanns, Jakob Mauss, Dietmar Neumerkel, et al. Functional mockup interface 2.0: The standard for tool independent exchange of simulation models. In *Proceedings of the 9th International MODELICA Conference; September 3-5; 2012; Munich; Germany*, number 076, pages 173–184. Linköping University Electronic Press, 2012.

[24] Peter A Cundall. A computer model for simulating progressive large scale movements in blocky rock systems. In *Proceedings Symposium Int. Soc. Rock Mech (ISRM)*, volume 1, pages 8–11, Nancy Metz, 1971.

[25] Christoph Kloss and Christoph Goniva. Liggghts–open source discrete element simulations of granular materials based on lammps. *Supplemental Proceedings: Materials Fabrication, Properties, Characterization, and Modeling, Volume 2*, pages 781–788, 2011.

[26] Christian Richter. A new approach for integrating discrete element method into component-oriented system simulations. In *ASIM 2016 - 23. Symposium Simulationstechnik 07.-09.09.2016. Zusammenfassung der Beiträge*, pages 91–97, HTW Dresden, 2016.

[27] Günther Kunze, Andre Katterfeld, Christian Richter, Hendrik Otto, and Christian Schubert. Plattform- und softwareunabhängige simulation der erdstoff-maschine interaktion. In *5. Fachtagung Baumaschinentechnik*, Dresden, 2012.

[28] Tobias Bellmann. Interactive simulations and advanced visualization with modelica. In *Proceedings of the 7th International Modelica Conference; Como; Italy; 20-22 September 2009*, number 043, pages 541–550. Linköping University Electronic Press, 2009.