Mobility-as-a-Service: A Distributed Real-Time Simulation with Carrera Slot-Cars

Daniel Richter, Andreas Grapentin and Andreas Polze
Operating Systems and Middleware Group
Hasso Plattner Institute at University of Potsdam
P.O.Box 90 04 60, D-14440 Potsdam, Germany
Email: {daniel.richter, andreas.grapentin, andreas.polze}@hpi.uni-potsdam.de

***Abstract*—*Mobility-as-a-Service* (MaaS) describes a class of applications where traditional real-time control systems are enhanced by backbone services accessed via the mobile Internet. In order to implement MaaS, new architectures for multi-stage real-time systems with several layers of control loops have to be implemented. Using approaches such as analytic redundancy, hard real-time control loops are extended with software-defined sensors that deliver data with soft real-time semantics.**

We describe a real-time control experiment that has been implemented in our *Distributed Control Lab* with four stages – an extended digital Carrera race track (D132), custom built sensor/actuator boards, a control PC, and an outer control loop established via web services – and present a timing analysis. Our architecture allows for decoupling of hard real-time processing on embedded control units and soft real-time data acquisition on the outer layers.

I. Introduction

The complexity of today's automobile hardware and especially software is steadily growing. Simultaneously, besides active safety features or entertainment systems manufacturers tend to implement "always online" solutions, where modern cars constantly are connected to services running on data-center infrastructure rather than on the car's IT systems.

The Internet and the car will no longer be separated – vehicles will be able to communicate with the outside world. This will also enable entirely new business models – such as apps and services for cars [1], [2]. For example, when the car is approaching the navigation destination a request for current parking information could be sent to a service which queries the most appropriate parking locations based on current availability, fees, and user-customized settings. Such services could also provide recommendations about special offers at gas stations, best parking deals or offers at shops along the route [3]. For example for BMW's new electrical i3 and i8 cars, the crucial electric range calculation is done on a remote host outside of the car. These backbone services require a continuous Internet connection whereas the car still can operate independently when the connection fails [4].

This shift in paradigms requires a reevaluation of the design of embedded real-time systems in automobiles that should be extended by non-real-time parts (like requests to cloud services). We propose a blueprint based on the cascaded interoperation of several layers of real-time functionality assisted by safety controllers.

II. Architecture for MaaS-enabled Applications

With embedded real-time systems connected to remote non-real-time services it is unavoidable that deadlines will be missed under certain circumstances. Wide-distance communication channels with protocols such as HTTP cannot guarantee compliance with any real-time constraints. To deal with that, we propose a cascade of different stages each with own safety controllers.

The idea is to create a chain of *cascade stages* (see figure 1) beginning with the one that requires the highest level of real-time constraints (respectively has the most stringent deadlines). The more stages are added, the less real-time constraints can be fulfilled – on the other hand the ability to implement more complex logic and integrate other services will grow.

A *cascade stage* consists of control logic to process requests, *communication channels* to adjacent stages, and a *safety controller*. The cascade stage is responsible to respond to requests of the previous stage within a certain deadline. To process requests, a stage usually depends on results delivered by a following stage – whereas the deadlines for subsequent stages are less stringent than the current one. This leads to the effect that a dependent stage may not deliver necessary results before the current deadline – in those cases a stage relies on its safety controller. The safety controller is responsible for providing data that does not necessarily have the best quality but is sufficient to provide the specified service.

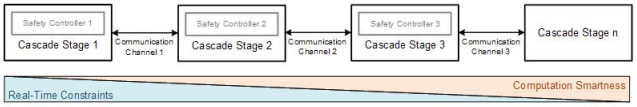

Fig. 1. Our architecture blueprint for the cascaded interoperation of several layers of real-time functionality assisted by safety controllers.

With this architecture it is possible to extend closed real-time systems with parts that are not necessarily capable to meet strict real-time constraints – for example best-effort components meet deadlines in most cases but do not guarantee them. In comparison to a conventional architecture with only one safety controller, the overall result will not degrade fully, even if outer stages miss their deadline. In addition, the one stage that cannot achieve the required real-time constraints can dynamically alternate among different implementations or

1555-0885/15 $31.00 © 2015 IEEE
DOI 10.1109/ISORC.2015.19

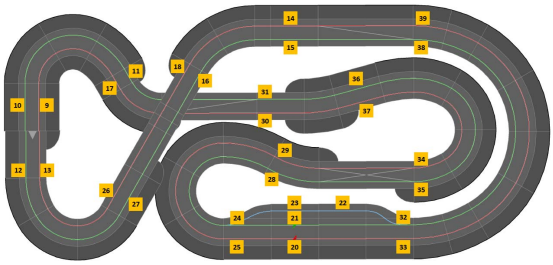

Fig. 2. The structure of our slot-car racetrack. We used an off-the-shelf construction kit and extended it with multiple position sensors (some of them indicated by yellow boxes).

service providers (i.e. applying multi-version programming). Therefore, our layered architecture with gradually decreasing real-time constraints can deal with variable deadline misses and will result in better overall behavior of the entire control system than a solution with only one safety controller operating on a single fixed deadline.

III. Implementation and Simulation

We have implemented our Mobility-as-a-Service system architecture in a distributed real-time system controlling a digital Carrera D132 race track consisting four stages.

The four layers of our system have different real-time requirements.

A. Stage 1 - Race-Track Control Unit

The inner-most stage consists of an off-the-shelf *Carrera Digital 132* race-track including the so called "Carrera control unit" that handles the actual control of the slot-cars by modulating a digital signal on the traction current. With D132, up to eight cars can be controlled by a control unit. The slot-cars are responsive to commands for setting the speed, breaking, and trigger turnouts. All cars have an inbuilt LED that sends out pulsed light including their current controller ID number (0-7). We have extended the race-track with several additional position sensors (see figure 2) based on phototransistors type SFH309FA.

Communication Channel: The control unit is connected to the *sensor/actuator board* via an RS232 interface as well as four RJ-12 interfaces and accepts standard *Carrera D132 protocol* commands. A data packet for a car contains the car's controller ID, the speed step, and a track change indicator flag. All position sensors are directly connected to the sensor/actuator board and transfer the raw analogue measurement signal.

Safety Controller: If there are no new commands submitted by the next cascade stage (sensor/actuator board) within 75 ms, the control unit will repeat creation of command packets based the last signal it received.

B. Stage 2 - Sensor/Actuator Board

The second stage is represented by a custom made circuit board including microcontroller firmware (written in C) that both simulates handheld controllers (responsible for speed, brakes, and turn switch points) as well as analyzes sensor signals. It has analogue input interfaces for sensor signals, an Atmel XMEGA 128 A3 microcontroller, RJ-12 outputs for handheld controller, and a FTDI FT232R USB UART. Our extension to the Carrera D132 system consists of 30 additional phototransistors that are placed along the course and generate position information for cars, that are decoded by our microcontroller firmware, encoded in a custom USB protocol, and handed over to the USB chip. Since position sensors are placed with certain distances, the control algorithm has certain "blind spots".

The microcontroller retrieves commands submitted via USB and translates them into the D132 protocol to both simulate handheld controllers (responsible for speed, brakes, and turn switch points) connected to the control unit as well as to send specific configuration commands (such as general speed limit, intensity of brakes, or fuel indicators). A safety controller function will always be able to stop (heavily decelerate) a car. The USB unit also is used to receive car control commands via the custom USB protocol that are translated to the Carrera D132 protocol and transfered to the Carrera control unit.

The sensor/actuator board itself has very strict real-time constraints: Non-observed signals from position sensors could lead to wrong assumptions about the position and speed of a car – which could result in a crash or an ejection of a car out of the slot-lane. Therefore, it is crucial to have interrupt service routines as short as possible. So, as much logic as possible is executed asynchronously with a lower priority than e.g. the edge detection for sensor signals.

Communication Channel: Stage 2 is connected to stage 3 (PC application) via USB. All data is encoded in a custom USB format. Because of internal buffering and the characteristics of the USB protocol this data transmission can neither be considered as real-time capable nor provide strict timing guarantees.

Safety Controller: The main goal for stage 2 is to keep all cars within a safe state. When a following stage does not deliver any control command within 150 ms (= two 75 ms periods), the microcontroller will send commands to stop all cars.

C. Stage 3 - PC Application

The control PC runs a control program that takes track layout information into account and computes trajectories for a single car. Based on the trajectory, control commands are sent to the microcontroller. Stage 3 runs an algorithm that is able to steer a car safely along the racetrack. However, it is not able to completely avoid collisions among cars as it only operates on snapshots of the racetrack (i.e. position sensors) and does not take into account potentially conflicting driving strategies.

As cascade stage 3 is a .NET application (written in C#) running on commodity PC hardware, no real-time guarantees are given on that stage.

Communication Channel: The communication to the last stage is implemented through HTTP / TCP/IP connections.

Because of arbitrary obstacles such as network latency, rerouting, service maintenance and updates, or resource scarcity no guaranteed response times can be specified.

Safety Controller: If following stages miss their deadlines and do not deliver any results, stage 3 will fall back safe strategy to steer all cars in a way that they will not derail and the possibility to collide is minimized. Without additional input, the safety controller basically will keep the car moving at a steady, slow speed.

D. Stage 4 - Cloud Service

The outer-most layer of our system is implemented by a set of web services. The services implement a number of multi-car driving strategies. They use time-distance-extrapolation for the cars to create a global set of car trajectories. Conflicting trajectories will be detected and control signals sent back to the control PC. In addition, the global trajectory information can be used to implement driving strategies such as "convoy" or "safety car". These services may take into account additional information about the environment (such as road conditions).

IV. Timing Analysis

The loose coupling of the cascade stages of the described architecture allows for the separate timing analysis of each stage with respect to the deadlines of its communication, as the safety controllers are designed to prevent timing failures from propagating to the stages below.

A. Stage 2 - Sensor / Actuator Board

The real-time capabilities provided by the microcontroller and the algorithms used in the Sensor / Actuator Board allowed the evaluation of its timing behaviour by producing a worst-case scenario for each component and measuring the execution times using an oszilloscope.

The track signal of the Carrera race track implements a Manchester Code that can produce a signal edge every $50\mu s$. Processing the edge in the microcontroller takes $2\mu s$. Additionally, the board monitors 4 position sensor ports each of which can be connected to up to 8 position sensors. The race cars on the track contain an LED that produces an edge on a position sensor signal every $k \cdot 32\mu s$ whith the car id $k \in [1, 8]$. An edge on a sensor signal triggers a procedure in the microcontroller that evaluates all sensors connected to the shared port that registered the edge. This process takes $5\mu s$. Given simultaneous edges on each monitored signal, the execution of the race track signal processing routine can not be delayed more than $20\mu s$ and each of the position sensor signal processing routine can not be delayed more than $17\mu s$. Both of these worst case delays are smaller than the time between possible signal changes.

With a lower priority, the microcontroller implements a main loop to produce USB packets to send to the next stage. This process takes $151\mu s$ and must be completed every *7.5s* to avoid missing a packet of the race track signal. Again, given continuous, simultaneous edges of maximum frequency on all signals, the system utilization produced by the signal handling routines is $\frac{2\mu s}{50\mu s} + \frac{20\mu s}{32\mu s} = 66.5\%$. Based on this, one iteration of the main loop can not exceed $\frac{151\mu s}{1-0.665} = 450.75\mu s$ which means that no deadlines may be missed even in the unrealistic worst case scenario we constructed. The actual system utilization during execution is much lower as far fewer position sensor edges are produced.

B. Stage 3 - PC Application

In contrast to the microcontroller used in stage 2, the PC implementing the control logic of this stage does not provide sufficient real time capabilities to allow for a stringent timing analysis. Instead, we estimate the probabilities of missed deadlines by measuring the runtime of the control logic and approximating the measurements as a normal distribution.

The PC hosting the control application contains an *Intel Core i5 3320M 2.6GHz* and uses the *Windows 7 Enterprise SP1* operating system. The controller logic is implemented in *.NET/C#*. We observed that the cpu utilization of the test system never exceeded *5%* during the tests. A histogram of the runtime measurements is presented in figure 3

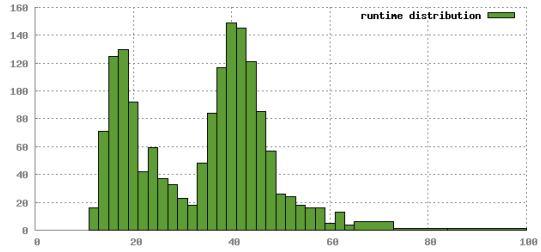

Fig. 3. The runtime measurements of the C# application per packet in μs with a histogram of $2\mu s$ granularity.

These measurements are approximated by a normal distribution of mean $\mu = 34\mu s$ and derivation $\sigma = 14\mu s$. Korver et.al. have evaluated the timing behaviour of USB connections [5] and have concluded that the communication overhead is approximately normally distributed and that for sufficietly small packets, the distribution of inbound connections can be represented as $\mu_I = 12ms; \sigma_I = 1.5ms$ and the distribution for outbound connections can be represented as $\mu_O = 6ms; \sigma_O = 0.1ms$.

Based on these distributions, the probability of missing the deadline of *75ms* between expected control updates is

$$P(X > 75ms) = \frac{1}{2} erf\left(\frac{\mu_{total} - 75ms}{\sigma_{total} \cdot \sqrt{2}}\right) \approx 7 \cdot 10^{-274}$$

.

C. Stage 4 - Cloud Service

While web services provided on the internet via HTTP often answer within a *50ms* window, the timing behaviour can not be guaranteed, and a probabilistic evaluation is difficult because of variances between separate web services. However, the services are not necessary for safe operation of the described system and only provide improved quality of the control algorithms. A missed deadline in this stage results in

a degraded system state where the safety controller in stage 3 needs to control the race cars with a slightly limited world view.

V. Related Work

The *Simplex Architecture* [6], [7] is a well-known approach to increase a system's reliability with the help of analytic redundancy. The idea is to have a safety controller whose logic is simple, well understood, and dependable and that can take over the control in cases where the system's state tends to leave a predefined safety envelope. While within the safety envelope, a advanced controller can deliver results in a higher quality than the safety controller, whose sole responsibility is to bring the system back into safe state. Within previous work, the authors have studied the integration of CORBA middleware with real-time control systems following the Simplex approach [8].

In contrast to our proposal, the Simplex architecture does not consider cascaded control stages. With Simplex, one has to choose between results either of the advanced controller or the safety controller. Our solution allows to also use slightly degraded results that are less valuable than the one of the high quality stage (advanced controller) but are better than results out of the inner-most stage (safety controller).

Mobile Cloud Computing [9] and *Mobile Code Offload* [10]–[12] allows to use cloud resources out of mobile devices. The goal is to optimize the resource usage within an embedded system so that resource intensive tasks are executed on a cloud infrastructure.

While many mobile cloud computing and mobile code offloading solutions require a permanent connection to a cloud system (always on) respectively only consist of independent mobile component and cloud service, our solution is capable to deal with connection losses and can adapt its service quality to changing environmental conditions.

The *Distributed Control Lab* (DCL) is a remote laboratory setup the author's institution that is being used to teach real-time and embedded systems classes. DCL hosts a variety of experiments [13], [14].

VI. Conclusions

Within this paper, we have presented a real-time control experiment that has been implemented in our *Distributed Control Lab*. We have described details of our implementation and presented a timing analysis. Our architecture employs the concept of cascaded processing stages (a layered approach) and allows for decoupling of hard real-time processing on embedded control units and soft real-time data acquisition on the outer layers.

Our lab experiment uses Web service technology to integrate non-real-time (cloud) services to implement multi-car driving scenarios. This solution is one example for the more general concept of "software defined sensors" that may act as extension to classical control systems.

From our experiment, we draw two main conclusions: (1) compute and network capacity of today's COTS hardware will often allow to meet real-time deadlines even when operating on a best-effort model and (2) the concept of software-defined sensors – in conjunction with the model of Analytic Redundancy – will allow for extensibility while still guaranteeing fail-safe behavior of the control system.

Future work will need to focus on security aspects. While being out of scope of this paper, the question of how to operate on faulty (compromised) data values from software-defined sensors needs to be taken into account in future work.

Acknowledgments

We would like to thank Uwe Hentschel for constructing the sensor and actuator managements boards as well as Björn Groneberg and Florian Zimmerman for their effort with measurements.

References

[1] "T-systems connected car vision: Driving in the clouds," *Best Practice - The T-Systems customer magazine*, no. Issue 03-2014, Mar. 2010. [Online]. Available: http://www.t-systems.com/umn/uti/765336_2/blobBinary/Best-Practice-issue_03-2010.pdf

[2] "Kommerzielle dienste für connected cars – winfwiki (in german)." [Online]. Available: http://winfwiki.wi-fom.de/index.php/Kommerzielle_Dienste_f%C3%BCr_Connected_Cars

[3] "BMW and SAP partner on connected car - m2m magazine." [Online]. Available: http://www.machinetomachinemagazine.com/2014/01/23/bmw-and-sap-partner-on-connected-car/

[4] "BMW copy - connected car report." [Online]. Available: http://www.connectedcar.org.uk/bmw-copy/4586820861

[5] N. Korver, "Adequacy of the universal serial bus for real-time systems," pp. 041 802–36, 2003. [Online]. Available: http://doc.utwente.nl/56344/

[6] T. L. Crenshaw, E. Gunter, C. L. Robinson, L. Sha, and P. R. Kumar, "The simplex reference model: Limiting fault-propagation due to unreliable components in cyber-physical system architectures," in *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*. IEEE, 2007, pp. 400–412.

[7] L. Sha, "Using simplicity to control complexity," *IEEE Software*, vol. 18, no. 4, pp. 20–28, 2001.

[8] A. Polze, J. Schwarz, K. Wehner, and L. Sha, "Integration of CORBA services with a dynamic real-time architecture," in *Real-Time Technology and Applications Symposium, 2000. RTAS 2000. Proceedings. Sixth IEEE*. IEEE, 2000, pp. 198–206.

[9] P. Bahl, R. Y. Han, L. E. Li, and M. Satyanarayanan, "Advancing the state of mobile cloud computing," in *Proceedings of the Third ACM Workshop on Mobile Cloud Computing and Services*, ser. MCS '12. New York, NY, USA: ACM, 2012, pp. 21–28.

[10] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2012, pp. 945–953.

[11] H. R. Flores Macario and S. Srirama, "Adaptive code offloading for mobile cloud applications: Exploiting fuzzy sets and evidence-based learning," in *Proceeding of the fourth ACM workshop on Mobile cloud computing and services*. ACM, 2013, pp. 9–16.

[12] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless communications and mobile computing*, vol. 13, no. 18, pp. 1587–1611, 2013.

[13] A. Rasche, B. Rabe, P. Tröger, and A. Polze, "Distributed control lab." in *VIRTUAL-LAB*. Citeseer, 2004, pp. 150–159.

[14] A. Rasche, P. Troger, M. Dirska, and A. Polze, "Foucault's pendulum in the distributed control lab," in *Object-Oriented Real-Time Dependable Systems, 2003. WORDS 2003 Fall. The Ninth IEEE International Workshop on*. IEEE, 2003, pp. 299–299.