RTSS@WORK 2019

Program Chair

- Song Han, University of Connecticut, USA

Program Committee

- Jingtong Hu, University of Pittsburgh, USA
- Hyoseung Kim, University of California Riverside, USA
- Jae W. Lee, Seoul National University, South Korea
- Shahriar Nirjon, UNC Chapel Hill, USA
- Selma Saidi, Hamburg University of Technology, Germany
- Rui Tan, Nanyang Technological University, Singapore
- Junlong Zhou, Nanjing University of Science and Technology, China

Message from the Program Chair

We cordially welcome you to RTSS@Work, the Open Demo Session of the 40th IEEE Real-Time Systems Symposium (RTSS).

The goal of RTSS@Work is to provide a platform for researchers to present prototypes, tools, simulators, and systems, that extend the state of the art in real-time technologies and techniques. This session augments the traditional forum by enabling presenters to demonstrate working systems, thereby allowing them to directly engage with the audience, generate interest in new research topics, and encourage wider adoption of common frameworks. This year, the Program Committee has selected eight high-quality demonstrations from diverse research areas and application domains.

We would like to thank all those who have contributed in making an excellent program for RTSS@Work 2019. We thank all the reviewers for their hard work in providing valuable feedback to the authors, and the organisation of RTSS 2019 for including this demo session in the symposium. Finally, we would like to thank all the authors for their fine contributions.

Song Han, University of Connecticut, USA

RTSS@Work 2019 Program Chair

Contents

MoFi: Environment-Independent Device-Free Human Motion Detection via WiFi

Xi Chen, Hang Li, Chenyi Zhou, Xue Liu, and Gregory Dudek
Samsung AI Center, Montreal, Quebec, Canada

***Abstract*—Detecting human motion using standard Wi-Fi signals in a device-free manner is a promising yet challenging task. The state-of-the-art systems passively collect the Wi-Fi Channel State Information (CSI), and train a model to extract motion information from the temporal changes in CSI. However, these systems are usually overfitted by the environment specific components of CSI, making hard to adapt to environment changes. To address this issue, we propose MoFi, which automatically filters out the impact of environment related features of CSI and adapts itself to a new or changed environment.**

I. Introduction

Device-free human motion detection is the process of detecting whether there is a human moving in an area of interest, without the requirement that they carry any special hardware. This is a fundamental functionality that can leverage a wide range of applications, such as security monitoring, smart device wake-up, automatic light control, and appliance automation. Compared with camera or LiDAR based systems, Wi-Fi based systems have several distinct benefits, including a ubiquitous coverage due to the widespread nature of Wi-Fi infrastructure, an ability to detect through walls/darkness provided by the signal propagation, the preservation of privacy and the low cost. The state-of-the-art Wi-Fi based systems (e.g., Widar [1]) analyze temporal changes in Wi-Fi Channel State Information (CSI) to detect human motion, and achieve promising performance in a controlled environment.

Those systems, however, have some notable limitations. The Wi-Fi signals propagating through an indoor environment embed a significant amount of environment specific information in the CSI. If the environment changes, the motion detection system previously trained and tuned will be corrupted.

To address this issue, we propose MoFi, an environment-independent device-free human motion detection system. MoFi **adaptively** adjust itself to different environments by automatically removing the impact of environment-dependent components in CSI. Moreover, unlike existing environment-adversarial systems, such as AutoFi [2] and EI [3], MoFi is a **plug-and-play** solution requiring no training. In addition, MoFi is able to meet the deadlines of periodic detection tasks.

II. Design of MoFi

A. Data Collector

Consider an area covered by the Wi-Fi signals between an Access Point (AP) and a commercial off-the-shelf (COTS) device. Suppose there are totally N_S Wi-Fi spatial streams, and the Wi-Fi channel is divided into N_C frequency subcarriers. MoFi is deployed on the Wi-Fi device, passively listening to the Wi-Fi traffic. For every received packet, the Wi-Fi chip on the COTS device extracts one CSI reading, which is an estimate of channel responses for all N_C subcarriers on all the N_S streams. Let $h(i,j)$ denote the complex CSI value of the ith subcarrier on the jth streams. Then the CSI magnitudes of the jth streams can be expressed as a vector $\mathbf{h}_j$ as $\mathbf{h}_j = [|h(1,j)|, |h(2,j)|, \cdots, |h(N_C,j)|]$.

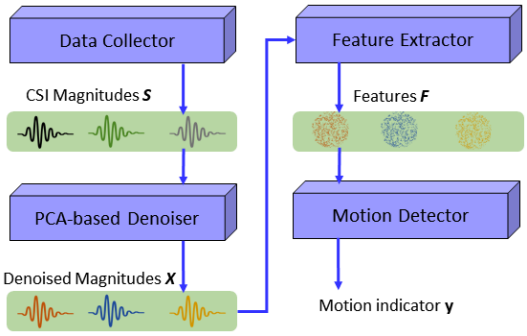

Fig. 1. Overview of MoFi

We then normalize the magnitudes of CSI for each stream into a range of $(0, 1]$. The normalized CSI magnitude vector is denoted as $\bar{\mathbf{h}}_j$. We concatenate these normalized vectors from N_S streams to create an aggregated CSI magnitude vector $\mathbf{s}_k$ for packet k as $\mathbf{s}_k = [\bar{\mathbf{h}}_1, \bar{\mathbf{h}}_2, \cdots, \bar{\mathbf{h}}_{N_S}]$. This $\mathbf{s}_k$ is then stored in a buffer of size N_{B1}. Once full, the buffer outputs an $N_C N_S \times N_{B1}$ dimensional CSI magnitude matrix $\mathbf{S}$ as

$$\mathbf{S}_k = [\mathbf{s}_k^T, \mathbf{s}_{k+1}^T, \cdots, \mathbf{s}_{k+N_{B1}-1}^T]^T. \quad (1)$$

After this,the oldest magnitude vector is discarded.

B. PCA-based Denoiser

COTS Wi-Fi APs and devices usually implement adaptive power control and Adaptive Modulation and Coding (AMC), which induce abrupt CSI shifts. To remove them, we employ a Principal Component Analysis (PCA) noise removal process. This denoiser first computes PCA on $\mathbf{S}_k$ to achieve an $N_C N_S \times N_P$ dimensional matrix $\mathbf{\Theta}_k$ containing the top N_P principal components (PCs), and a corresponding PC score vector $\mathbf{v}_k$. We then set the first PC (which captures the majority of the abrupt noise [4]) to a zero vector, and obtain a denoised PC matrix $\hat{\mathbf{\Theta}}_k$. Also, we set the first PC score to zero, and achieve a denoised PC score vector as $\hat{\mathbf{v}}_k$. We use them to reconstruct a denoised CSI magnitude matrix $\mathbf{X}_k$ as

$$\mathbf{X}_K = \hat{\mathbf{v}}_k \times \hat{\mathbf{\Theta}}_k. \quad (2)$$

Note that $\mathbf{X}_K$ is not the project of $\mathbf{S}_k$ onto the PCs, but a CSI magnitude matrix reconstructed after removing the first PC.

C. Feature Extractor

Each row of an $N_C N_S \times N_{B1}$ dimensional denoised matrix $\mathbf{X}_k$ is a time sequence of length N_{B1} (from the kth packet to the $(k+N_{B1}-1)$th packet). We calculate the variance of each row to achieve a variance vector $\mathbf{v}_k$ of length $N_C N_S$ as $\mathbf{v}_k = [v_1, \cdots, v_{N_C}, v_{N_C+1}, \cdots, v_{2N_C}, \cdots, v_{N_C N_S}]$. Variance solely is not a robust indicator of human motions, as it also contains unknown changes caused by (residual) random noise. In this paper, we propose to use the Variance of Variance (VOV) as a robust feature, which eliminates the random variation from noise and focuses (almost) purely on human motion. Concretely, we setup another buffer of size N_{B2} to store the variance vectors. Once the buffer is full, it outputs an $N_C N_S \times N_{B2}$ variance matrix $\mathbf{V}_k$ as $\mathbf{V}_k = [\mathbf{v}_k^T, \mathbf{v}_{k+1}^T, \cdots, \mathbf{v}_{k+N_{B2}-1}^T]^T$, and discards the oldest element from the buffer. For each row in $\mathbf{V}_k$, we calculate the variance, and achieve a VOV vector $\mathbf{F}_k$ of length $N_C N_S$ as

$$\mathbf{F}_k = [f_1, \cdots, f_{N_C}, f_{N_C+1}, \cdots, f_{2N_C}, \cdots, f_{N_C N_S}]. \quad (3)$$

D. Motion Detector

In this paper, we adopt an environment-independent plug-and-play detector, which requires no training and provides rapid response. Given a $\mathbf{F}_k$, we calculate per-stream minimums to obtain a vector $\mathbf{z}_k$ as $\mathbf{z}_k = [z_1^{(k)}, z_2^{(k)}, \cdots, z_{N_S}^{(k)}]$, where $z_n^{(k)} = \min(f_{(n-1)N_C+1}, \cdots, f_{nN_C})$ is the minimum VOV in the nth stream from the kth feature vector. We then compare this minimum vector to a pre-defined threshold vector $\gamma = [\gamma_1, \cdots, \gamma_{N_S}]$. If all the elements in $\mathbf{z}_k$ is larger than the corresponding one in γ, then MoFi outputs an motion indicator

$$y_k = \begin{cases} 1 & \text{if } \forall i, z_i^{(k)} > \gamma_i, \\ 0 & \text{otherwise}, \end{cases} \quad (4)$$

and store it in an indicator buffer of size N_{B3}. If the buffer is full, the oldest element is discarded.

To support motion detection tasks with a period of T seconds, MoFi has to gather at least N_{B2} packets every T seconds (suppose the N_{B1} buffer is full). To this end, the Wi-Fi traffic must be faster than $\lceil N_{B2}/T \rceil$ pps. If the on-the-air Wi-Fi traffic is slower than this speed, MoFi will send out extra pings to boost the traffic. In this way, MoFi can output at least one y_k every T seconds. Suppose MoFi already collected $N_D \leq N_{B3}$ motion indicators $[y_k, \cdots, y_{k+N_D-1}]$, when a periodic task deadline is δ seconds ($\delta \ll T$) away. At this moment, MoFi outputs one detection result as

$$Y_{motion} = \begin{cases} \text{True}, & \text{if } \sum_{i=0}^{D-1} y_{k+i} > 0, \\ \text{False}, & \text{otherwise}. \end{cases} \quad (5)$$

III. Experiments

In our experiments, we used a TP-Link AC1750 router as the TX, and a Dell Latitude E7440 laptop with Intel 5300 Wi-Fi NIC as the RX. We installed Linux CSI Tool [5] to extract CSI from the Intel NIC on the RX. MoFi was also implemented on the RX side. Suppose an upper layer application poses a periodic motion detection task with a period of 1 second. Unlike previous systems that require heavy Wi-Fi transmissions (e.g., [1], [3], [4]), MoFi sets the Wi-Fi traffic rate between TX and RX as low as 20 pings per second. The buffer sizes are set to $N_{B1} = 20$, $N_{B2} = 20$, and $N_{B3} = 40$, respectively. Empirically, we found that $\forall i, \gamma_i = 0.0001$ is a robust setting under different environments.

Fig. 2. Example experiment setup in an office room

We conducted different experiments to evaluate MoFi under different environment changes. False Positive Ratio (FPR) and False Negative Ratio (FNR) were used as the performance metrics. We compared MoFi with the motion detection module of AutoFi. Both systems were setup under the same environment. Later, we changed the environment to see how well they adapted to different changes. For each environment change, we recorded data for roughly 2 hours, which corresponds to $144,000$ Wi-Fi packets and $7,200$ detection outputs. Table I summarizes the comparison. We see that the VOV-based MoFi outperformed the variance-based AutoFi. Moreover, MoFi maintained its good performance in changed or even new environments without calibration. The FNR remained 0.0% while FPR was as low as 0.014%.

TABLE I. Motion Detection Results

Env. Changes	MoFi (FPR;FNR)	AutoFi (FPR;FNR)
No change	0.000%; 0.000%	2.125%; 9.681%
Turned on microwave	0.000%; 0.000%	2.069%; 9.986%
Different users	0.000%; 0.000%	2.986%; 11.361%
Moved the chairs	0.000%; 0.000%	6.902%; 13.986%
Opened the door	0.000%; 0.000%	7.069%; 14.375%
Rotated the router&laptop	0.000%; 0.000%	19.069%; 8.208%
Moved to another room	0.014%; 0.000%	17.291%; 13.208%

References

[1] Y. Zheng, Y. Zhang, K. Qian, G. Zhang, Y. Liu, C. Wu, and Z. Yang, "Zero-effort cross-domain gesture recognition with wi-fi," in *MobiSys*. ACM, 2019, pp. 313–325.

[2] X. Chen, C. Ma, M. Allegue, and X. Liu, "Taming the inconsistency of wi-fi fingerprints for device-free passive indoor localization," in *INFOCOM*. IEEE, 2017, pp. 1–9.

[3] W. Jiang, C. Miao, F. Ma, S. Yao, Y. Wang, Y. Yuan, H. Xue, C. Song, X. Ma, D. Koutsonikolas, W. Xu, and L. Su, "Towards environment independent device free human activity recognition," in *MobiCom*. ACM, 2018, pp. 289–304.

[4] K. Ali, A. X. Liu, W. Wang, and M. Shahzad, "Keystroke recognition using wifi signals," in *MobiCom*. ACM, 2015, pp. 90–102.

[5] D. Halperin, W. Hu, A. Sheth, and D. Wetherall, "Tool release: Gathering 802.11n traces with channel state information," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 1, pp. 53–53, 2011.

Polygraph Tool Suite: Configuration and Conformity Validation for Data Flow Based Real-Time Systems

Shuai Li, Matteo Morelli, Ansgar Radermacher, Jérémie Tatibouët, Pauline Deville, Arnault Lapitre, Sébastien Gérard, Chokri Mraidha
CEA LIST
Paris-Saclay, France
<first-name>.<last-name>@cea.fr

***Abstract*—With the shift towards software centric architectures, automotive OEMs need formal models and tools to tackle new integration challenges that severely increase the complexity of real-time system synthesis. Polygraph Tool Suite proposes a new Synchronous Data Flow model for functional architecture analysis. It provides static model validation, model simulation, schedulability analysis, and platform-specific configuration file generation. Post-deployment, runtime conformity to the specification is automatically validated.**

Keywords—real-time scheduling, schedulability analysis, conformity validation, synchronous data flow, automotive

I. Introduction

Motivated by the integration of autonomous driving and connected cars, there is a paradigm shift in the automotive industry towards platforms with more computing capability and software-centric architectures. As critical software becomes more prominent, engineers must tackle new challenges that increase the complexity of real-time system synthesis. Models and tools of the real-time embedded system community must be adopted in order to help OEMs integrate applications provided by tier 1 and 2 suppliers.

Polygraph Tool Suite (PTS) is a design tool that aims at helping software integration in real-time data flow oriented systems. The tool provides a modeler for Polygraph [8], a Model of Computation and Communication (MoCC) that extends Synchronous Data Flow (SDF) [1] with frequency arithmetic and adjustable communication rates (fractions). As a configuration tool, PTS helps develop consistent (e.g., no buffer overflows), alive (e.g., no deadlocks), and schedulable systems. As a conformity validation tool, PTS automatically checks correctness of communication rates and real-time properties of the runtime, w.r.t. the design model. A demo video, of an Advanced Driver Assistance System (ADAS), is available online [2].

Schedulability analysis is a major tool of PTS and a focus of this paper. Existing schedulability analysis tools [3, 4, 9] are not incompatible, but complementary with PTS. Indeed PTS uses classical real-time scheduling theory to analyze its SDF-based models, with platform allocations, as suggested by [5].

Compared to SDF-based tools [6], PTS is based on an extended SDF model. Furthermore, as mentioned, contrary to such tools, PTS analyzes real-time constraints not only through finding a cyclic behavior in the SDF system, but through classical real-time scheduling theory.

In this paper, PTS focuses on the automotive domain, therefore it is complementary to implementation platforms like AUTOSAR – a standard automotive software architecture – and its development tools [7]. Indeed, PTS has an underlying formal SDF model whose semantics can be integrated in the AUTOSAR meta-model. Unlike AUTOSAR tools that mostly target implementation design, PTS is at the Functional Architecture Analysis [10] level of abstraction. PTS also fills the gap between both levels of abstraction through AUTOSAR-specific ARXML generation. Compared to design tools based on Architecture Description Languages (ADL), such as AADL and EAST-ADL [10], PTS uses a formal MoCC that can be integrated in these ADLs.

The rest of the paper is organized as follows. In Section II, the Polygraph model is informally described. Section III showcases the PTS methodology and its tools. Section IV concludes the paper by proposing some future work.

II. Polygraph, an Extension of SDF

The underlying model of PTS is an extension of Synchronous Data Flow (SDF). The model and its theorems are published in [8]. In this paper, we give an informal description.

Polygraph adds frequency constraints, rational communication rates, and untimed actors that are not time-triggered. The latter lessens the synchronous constraints of classic SDF, which limits too much computing resources and is therefore not adapted to automotive systems where resources and costs must be optimized.

A Polygraph model is a graph with Actors connected by directed Channels. Actors represent tasks that consume some incoming data tokens, processes them, and produce some outgoing data tokens. The number of tokens, that an Actor needs for activation, is specified by its incoming Channels. Likewise for the number of tokens an Actor produces. An Actor can also be activated periodically. Actors have a Worst Case Execution Time (WCET), activation jitter, fixed-priority, and processor allocation.

A consistent Polygraph model has a cyclic behavior represented by Actors' activation vectors [1, 8]. Such a vector defines time slots in which Actors must activate and finish. The duration of a slot depends on the frequencies specified in the model. The total duration is the hyperperiod of Actors.

III. Polygraph Tool Suite Methodology and Tools

Within Polygraph Tool Suite, we have implemented a methodology automated as an Eclipse Cheat Sheet. In this paper

we describe the configuration and conformity validation tools, in which each iteration consists in (A) specifying a consistent functional architecture, thanks to formal modeling, static validation, and simulation in Polygraph, (B) proposing a schedulable task model, (C) configuring the software and implementing it, and finally (D) validating the implementation at runtime. The following sections describe each of these steps and their tools.

A. *Modeling, static validation, and simulation*

PTS offers a graphical modeling tool to design a system in the Polygraph formalism. The modeling language is implemented as a UML profile in the open source Papyrus modeler (http://eclipse.org/papyrus).

Consistency and liveness of Polygraph models can be statically validated during modeling, using adapted classic SDF theorems. An execution engine for Polygraph can simulate the model to evaluate functional correctness. The simulator integrates full debug functionalities (breakpoints, values).

B. *Schedulability analysis*

Actors have time constraints inferred from their activation vectors. They also have platform-specific annotations. In this paper, we assume partitioned multiprocessor platforms with fixed-priority scheduling policy. A real-time transaction task model [11] is automatically generated from the Polygraph model. The task model, with its time parameters such as period, offset, jitter, and deadline, expresses jobs representing Actor activation vectors. The task model is specified using the MARTE [10] modeling language. MARTE serves as a pivot for many schedulability analysis tools such as Time4Sys [9], MAST [4], and now PyCPA [3] through the work of this paper.

The task model is then translated into PyCPA Python scripts for compositional schedulability analysis. We adapted PyCPA for task models resulting from Polygraph, in order to lessen pessimism through precedence dependencies exploitation [11]. Worst Case Response Times (WCRT) are shown in Gantt charts, while processor load are presented as pie charts. The results are propagated back to the different models for traceability. Fig. 1 shows the schedulability analysis tool.

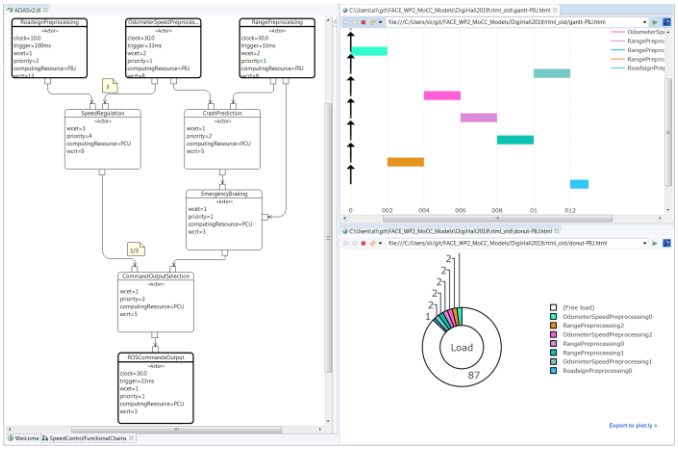

Fig. 1 Polygraph Tool Suite schedulability analysis: Polygraph model on the left, WCRT in Gantt chart, processor utilization in pie chart

C. *Configuration file generation*

For schedulable systems, platform-specific configuration files are generated for each Polygraph Actor. In this paper, we target ARXML files of AUTOSAR Adaptive. The ARXML files contain information about, e.g., interfacing and allocation. OEMs forward these files as specifications to their suppliers. Since the files are generated from valid system designs, such an approach limits specification errors and accelerates integration in a multi-supplier economic model.

D. *Conformity Validation*

Post-deployment, at runtime, incorrect Actor internal implementation by suppliers may violate the system constraints. PTS offers a runtime conformity validation tool.

Polygraph runtime monitors produce traces containing information on Actor activations, end of computation, and tokens exchanged. PTS detects deviations between traces and the formal design model. For debugging purposes, the tool is also able to replay and animate the traces in the model.

IV. Conclusion and Future Work

This paper presented Polygraph Tool Suite. The tool assists real-time software development by providing a methodology with formal SDF modeling and simulation, schedulability analysis, implementation configuration, and runtime conformity validation.

Currently Polygraph is being extended for hierarchical modeling. Furthermore, we will integrate the Polygraph MoCC into front-end ADLs such as EAST-ADL [10]. Although PTS is mostly built upon open-source software (Papyrus, PyCPA) some of its SDF tools, still closed-source, will be distributed in open-source in the future.

References

[1] E. A. Lee and D. G. Messerschmitt, “Static scheduling of synchronous data flow programs for digital signal processing,” in *IEEE Trans. Comput.*, vol. 36, no. 1, pp. 24-35, Jan, 1987.

[2] https://youtu.be/ZR2XDEqvCmA

[3] J. Diemer, P. Axer. and R. Ernst, “Compositional performance analysis in python with pycpa,” in *WATERS 2012*, Pisa, Italy.

[4] M.G. Harbour, J.G. García, J.P. Gutiérrez and J.D. Moyano, “Mast: Modeling and analysis suite for real time applications,” in *ECRTS 2001*, Delft, Netherlands.

[5] A. Singh, P. Ekberg, and S. Baruah, “Applying Real-Time Scheduling Theory to the Synchronous Data Flow Model of Computation,” in *ECRTS 2017*, Dagstuhl, Germany.

[6] J. Eker *et al.*, "Taming heterogeneity - the Ptolemy approach," in *Proceedings of the IEEE*, vol. 91, no. 1, pp. 127-144, Jan. 2003.

[7] U. Honekamp, "The Autosar XML Schema and Its Relevance for Autosar Tools," in *IEEE Software*, vol. 26, no. 4, pp. 73-76, July-Aug. 2009.

[8] P. Dubrulle, S. Louise, C. Gaston, N. Kosmatov, A. Lapitre, “A Data Flow Model with Frequency Arithmetic,” in *FASE 2019*, Prague, Czech Republic.

[9] L. Fejoz *et al.*, “Time4Sys – Integrating Timing Verification in your Engineering Practices,” in *RTSS@Work 2018*, Nashville, TN, USA.

[10] F. Mallet, M.A. Peraldi-Frati and C. André, “MARTE CCSL to execute East-ADL timing requirements,” in *ISORC 2009*, Tokyo, Japan.

[11] J.C. Palencia, and M.G. Harbour, “Exploiting Precedence Relations in Schedulability Analysis of Distributed Real-Time Systems,” in RTSS 1999, Phoenix, AZ, USA.

LiteOS for Intermittent Computing

Nan Guan
The Hong Kong Polytechnic University

Qiulin Chen
Huawei Technologies Co., Ltd.

***Abstract*—It is expected that trillions of IoT devices will be in use in near future. One fundamental challenge to deploy and maintain such a large number of devices is how to power them in an efficient, low-cost and sustainable manner. The rapid development of energy-harvesting hardware technology and ultra-low-power computing systems make it possible to build battery-less IoT devices that are completely powered by energy harvested from environment. A notable feature of such system is intermittent availability of computation resource, which raises significant challenges to the design of both software and hardware for such computing systems. To address such challenges, a new computing paradigm, namely Intermittent Computing, has been proposed and drawn increasing attentions in recent years. The key enabler for intermittent computing is the operating system, which provides fundamental services to support correct intermittent execution of applications under unpredictable power supply. In this paper we present LiteOS-Intermitter, a lightweight operating system to support intermittent computing based on Huawei LiteOS.**

I. Introduction

In recent years, there have been increasing requirements for ultra-low-power IoT devices. Advances in energy-harvesting technology have made it possible to build IoT devices that execute completely using energy harvested from the surrounding environment, so that the device life-time is not restricted by the battery. The computation in such systems is *intermittent* as the the energy is not always available to be harvested. Software running on an intermittent computing system executes until energy is depleted and the device browns out. When energy is again available, software resumes execution from some point in the history of its execution. In an energy-harvesting system, the power failures may occur very frequently (e.g., hundreds of time per second). Therefore, people must design the system in an intermittence-aware manner so that the it can make meaningful progress in the presence of frequent power failures.

Intermittent computing raises challenges on different software/hardware layers, especially to the operating system which provides essential abstraction and management of the hardware resource. Recently, many tiny operating systems and kernels have been developed to support intermittent computing, including Ratchet [3], Clank [6], HarvOS [1], Chain [2], Mayfly [4], Chinchilla [8], InK [5], etc. These systems can be classified into two categories: instruction-level check-pointing and task-based/transactional execution model. In this paper, we present **LiteOS-Intermitter**, a lightweight operating system to support intermittent computing based on Huawei LiteOS. Our system adopts the transactional execution model, and aims to provide a more comprehensive operating system support comparing with existing systems in the same category.

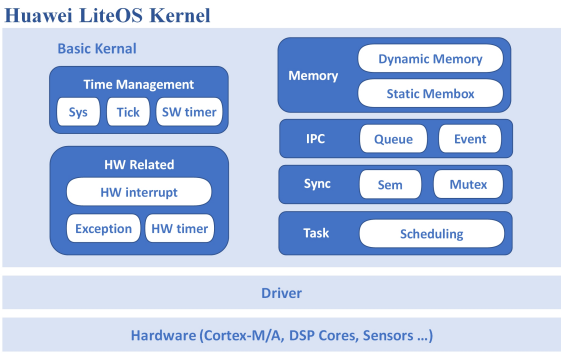

Fig. 1. Basic Kernel of Huawei LiteOS

II. Brief Overview of Huawei LiteOS

Huawei LiteOS is an open-source lightweight operating system designed for IoT devices, wearable devices and other power and time sensitive computing systems. The basic kernel size is smaller than 10K, and the response time is within 100 us on typical embedded processor platforms. The major component of Huawei LiteOS's basic kernel is shown in Figure 1. Upon the basic kernel, Huawei LiteOS provides a Sensor Framework to support low-delay, high-precision sensing by various Intelligent sensing algorithms, a Connectivity Engine to support a rich set of connectivity technologies (such as short-distance, LTE, and NB-IoT communication), and an Operating Engine to support a high-performance and lightweight VM based on JavaScript. More information about Huawei LiteOS can be found at [7].

III. LiteOS-Intermitter

We develop LiteOS-Intermitter based on the basic kernel of Huawei LiteOS to support intermittent execution of applications. LiteOS-Intermitter adopts the transactional execution model, modifies several components in the kernel and provides several new kernel services, and provide several analysis tools to facilitate the developer to develop correct and efficient intermittently executing applications.

A. Transactional Execution Model

Intermittent computing requires that once the system encounters power failure, the whole system must roll back to a previous execution point where system state is consistent (i.e., checkpoint). Existing checkpoint techniques typically save the whole system image at given time spots, the overhead of which is not acceptable for IoT devices.

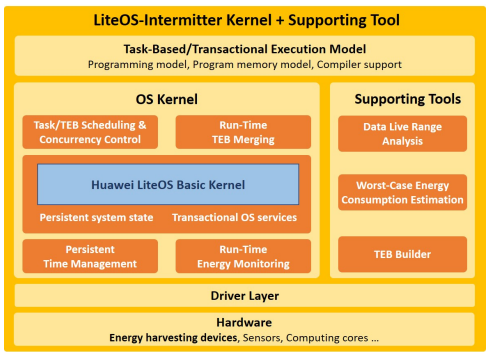

Fig. 2. The Major Components of LiteOS-Intermitter.

We adopt a transactional execution model for application development to solve the problem. In LiteOS, an application is programmed as a task (i.e., a process). In LiteOS-Intermitter, a task is composed of a series of atomic transactional execution blocks (TEB). As a TEB starts, related data are copied from NVM to RAM, and the new values are committed to NVM at the end of the TEB. If a TEB encounters power failure during its execution, the OS will restart from the entry of the un-finished TEB. As TEB construction is done at design time, further optimizations can be performed to reduce system overhead, such as data volume written to NVM and unnecessary task re-execution.

B. New Kernel Services

The state consistency must be guaranteed for not only the applications, but also the operating system itself. LiteOS-Intermitter includes the following features to support intermittent execution of the operating system.

- **Persistent system state**. Like in the execution of applications, when power failure occurs in an OS service, the system should roll back to a recent state-consistent point. This is achieved by persistently maintaining system variables on NVM.
- **Transnational OS execution**. Similar to TEBs, merely storing system variables on NVM will not guarantee correct execution. The update to system variables by an OS service must be implemented as a transaction. A new layer of operation is built on existing LiteOS service routines to make them run as transactions.

Moreover, the following functionalities are currently being revised/added to LiteOS to support intermittent execution.

- **Concurrency control**. To guarantee a TEB's atomic execution in the presence of preemption and interrupt handling, we adopt deferred concurrency control. If a preemption/interrupt will break a TEB's execution, we defer the response to such activities to the end of the TEB.
- **Run-time TEB merging** To reduce the overhead for storing/loading data to/from NVM at TEB boundaries, we implemented a run-time TEB merging mechanism. As long as the predicted energy is enough, the system will proactively merge multiple user-specified TEBs into a big TEB, or a big transaction, by which less data will be written to NVM.
- **Persistent time management**: Normal computing systems do not have to maintain persistent system time, since once the system restarts, all processes are destroyed and restarted. In intermittent systems, as a task (composed of multiple TEBs) restarts from an intermediate location, a consistent and persistent system time must be provided to the task. We provide a framework to allow system designers to be aware of the current time from external time sources. In cases where external time is not available, LiteOS-Intermitter provides an interface for the application developers to handle exceptions regarding lost of system time.
- **Run-time energy monitoring**. A key run-time service is to precisely monitor the available energy state to support dynamic TEB merging as long as there is enough energy. The main difference from predicting the energy percentage of a battery is that our energy monitoring module has to furthermore predict energy harvesting to precisely estimate future energy state for safe and efficient intermittent execution.

C. Analysis Tools

Currently, we are in the progress of developing a set of analysis tools to help the developers to build correct yet efficient applications on LiteOS-Intermitter.

- **Data live range analysis** will be conducted after the applications are developed. The analysis makes sure that data, which have to be stored to NVM on TEB boundaries, are not neglected by application developers in the declaration. On the other hand, the analysis will also help to find data that are not needed to be stored to NVM, and thus reduces state maintenance overhead.
- **Worst-case energy consumption estmiation** must be performed to ensure progress of the system. Although runtime energy monitoring is possible, off-line analysis of TEBs will provide strict guarantee to fit TEBs into given energy envelop.
- **TEB builder** automatically decompose the original task into TEBs (or big TEBs into smaller TEBs) based on the capability of the above two supporting functionalities.

REFERENCES

[1] Naveed Anwar Bhatti and Luca Mottola. Harvos: Efficient code instrumentation for transiently-powered embedded sensing. In *IPSN*, 2017.
[2] Alexei Colin and Brandon Lucia. Chain: tasks and channels for reliable intermittent programs. *OOPSLA*, 2016.
[3] Joel Van Der Woude etc. Intermittent computation without hardware support or programmer intervention. In *OSDI*, 2016.
[4] Josiah Hester etc. Timely execution on intermittently powered batteryless sensors. In *SenSys*, 2017.
[5] Kasim Yildirim etc. Ink: Reactive kernel for tiny batteryless sensors. SenSys, 2018.
[6] Matthew Hicks. Clank: Architectural support for intermittent computation. In *ISCA*, 2017.
[7] Huawei LiteOS. https://github.com/liteos.
[8] Kiwan Maeng and Brandon Lucia. Adaptive dynamic checkpointing for safe efficient intermittent computing. In *OSDI*, May 2018.

A Comprehensive Framework for Energy Management of Hard Real-time Networks-on-Chip

Thawra Kadeed, and Rolf Ernst
IDA, TU Braunschweig
Braunschweig, Germany
{kadeed|ernst}@ida.ing.tu-bs.de

***Abstract*—To cope with the ever-increasing complexity of multi processing architecture, Networks-on-chip (NoCs) are employed as a promising solution for Multiprocessor Systems-on-Chip (MPSoCs). In turn, NoCs' associated energy consumptions have immensely increased. Specifically, hard real-time Networks-on-chip must manifest limited energy consumption as reliability issues in such a shared resource jeopardize all system guarantees. In this paper, we propose a safe and efficient approach that allows global and online energy management under temporal guarantees, i.e., all deadlines of critical functions are met. The approach introduces a comprehensive framework to save energy on the NoC data layer through a control layer that contains multiple *Power-Aware Network Controllers (PANCs)*. *PANCs* explore and integrate multiple energy-savings schemes in the face of the diversity of energy dissipation sources. To safely apply the *PANCs* in hard real-time systems while meeting the deadlines, a formal worst-case timing analysis of the additional latency induced by the control layer is provided. This paper describes the comprehensive framework for NoC energy-savings through the *PANCs*, and illustrates the demonstration scenario.**

I. Introduction

Networks-on-chip (NoCs), as a scalable and modular interconnect, are employed as a promising solution for Multiprocessor Systems-on-Chip (MPSoCs). In such complex chips, NoC high energy consumption is a critical concern as it may lead to system reliability issues. NoCs contribute significantly to the energy consumption of total chip energy budget (e.g., SCORPIO research chip design [1]). In this paper, we investigate integrated energy control for NoC routers. Energy dissipation is mainly caused by dynamic and leakage energy sources. The diversity of energy sources calls for an integration of multiple energy-savings schemes, e.g., *Dynamic Voltage and Frequency Scaling* (*DVFS*), *Clock-Gating* (*CG*), and *Power-Gating* (*PG*).

Several NoC energy management schemes have been intensively researched. However, applying them to real-time NoC systems makes critical functions vulnerable, jeopardizing temporal guarantees. Thus, those schemes must guarantee the real-time constraints. *DVFS* reduces the NoC power consumption by adjusting the NoC frequency. A recent contribution in optimizing the power for hard real-time NoCs employing *DVFS* is done by [2]. *DVFS* is still limited to the circuit physical properties. That is, it cannot lower the frequency/voltage below the transistor limited bounds that are set during the design process. *CG* has a strong effect on energy-savings as the major dissipation of dynamic power is caused by the clock-tree that even increases the quiescent power (i.e., no load). The clock-tree power overhead is attributed to the activity on the clock pins of high-level non-clock gated synchronous elements, as introduced in [3]. *PG* schemes feature leakage power reduction. A recent work [4] provides a method for energy-savings employing *PG* when real-time systems are concerned. However, *PG* schemes are limited to solely leakage energy-savings.

Integrating the aforementioned energy-savings schemes is highly promising as it targets simultaneously different energy sources. Thus, we propose a comprehensive framework that provides global and dynamic NoC energy management using a control layer. The latter employs *power-aware network controllers (PANCs)*, which efficiently manage the NoC energy based on the current network utilization and the application real-time requirements. The *PANCs* feature multiple energy management schemes to enable significant energy-savings under temporal guarantees.

The major contribution of this paper is to explore the impact of the joint application of *CG*, *PG*, and *DVFS*, using a comprehensive framework for NoC energy control through the *PANCs*. Additionally, as we comply our approach with hard real-time requirements, an analysis framework to derive the worst-case timing overhead of our approach is provided in order to derive the safe applicability of the *PANCs*. To the best of our knowledge this is the first work that investigates integrating multiple energy-savings schemes, while simultaneously providing hard real-time temporal guarantees for NoCs deployments in safety critical domains (e.g., automotive).

II. Dynamic Energy Control Demonstration

In this paper, we propose a predictable and safe energy management approach for NoC routers, which extends the work of Kadeed and et al. [5] in order to emphasize the comprehensive NoC energy control framework. The approach is safe to be applied on hard real-time systems as it provides system temporal guarantees. Figure 1 illustrates the block diagram of the framework, employing *PANCs*. The demonstration at the conference will mainly showcase the blocks in the framework, highlighted in orange.

Additionally, we employ the control of a vehicle with assistance functions usecase [5], in order to demonstrate the impact of *PANC* in automotive domains. The experimental results indicate that *PANC* achieves 91.1% energy savings

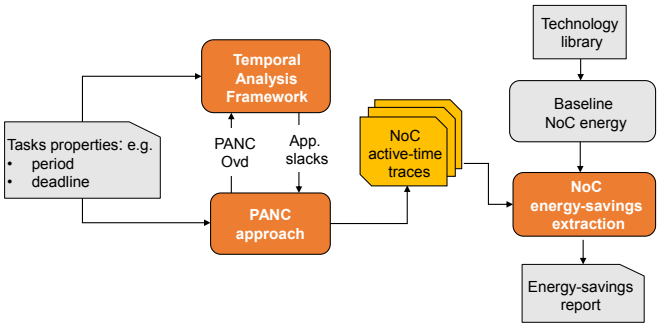

Fig. 1: Block diagram of the comprehensive framework for NoC energy management, employing *PANC*

compared with a baseline NoC (none of the energy-savings schemes has been applied), as detailed in [5]. To this end, the application tasks properties are provided first to *PANC* as an input. Moreover, tasks deadline slacks are also provided to *PANC* through the temporal analysis framework, which relies on *PANC* latency overhead (*PANC* Ovd) to compute the slacks. The deadline slacks $DSlack_i$ define the time budget between the task's deadline (D_i) and its worst-case traversal time, accounting for the *PANC* latency overhead (T_i^{PANC}). It is computed as follows:

$$DSlack_i = D_i - T_i^{PANC}, \quad (1)$$

The computation of T_i^{PANC} is performed through the temporal analysis framework. Thus, as we conform our approach to hard real-time systems where tasks deadlines should be met, *PANC* applies *CG*, *PG*, and *DVFS* on NoC routers upon tasks positive slacks. During run-time, *PANC* generates traces that reflect the observed NoC active times. That is, the trace contains varying time intervals under which the NoC router is on/off with the respective frequency. The NoC energy-savings extraction block relies on the observed traces and the baseline NoC energy in order to derive the NoC energy-savings figures. The baseline NoC energy is derived using ASIC design flow employing Synopses tool chain for both $65nm$ (UMCs) and $28nm$ (TSMCs) technologies [4], [5].

A. PANC approach

PANC is integrated in the OMNeT++ NoC simulator in order to control the NoC power. It implements a protocol-based synchronization to know the global state of the NoC, and applies this knowledge to save energy while keeping the system predictable [5]. Thus, once *PANC* receives a NoC request/release *(Req/Rel)* message from a sender, it goes through three steps to serve the message, as depicted in Figure 2. First, it arbitrates between the requests to serve the highest priority one. Second, it processes the request by making a decision of the most efficient energy modes of the routers. Eventually, it translates these decisions into commands to the respective actuators in the NoC system, and acknowledges the sender. Figure 2 depicts an abstract view of NoC employing *PANC* - integrating the energy-savings schemes (*CG*, *PG*, and *DVFS*) to efficiently control the NoC energy. *PANC* signals the clock generator with the required clock mode derived from the current NoC traffic. The clock generator then provides NoC with the respective clock frequency, leading to a frequency increase/decrease *(DVFS)*. Moreover, as soon as *PANC* detects an idleness state of certain routers, it signals them with *PCG_ON* messages (power/clock-gating on) in order to gate their power (*PG*: using the power switch), and their clock (*CG*: using clock-gating blocks).

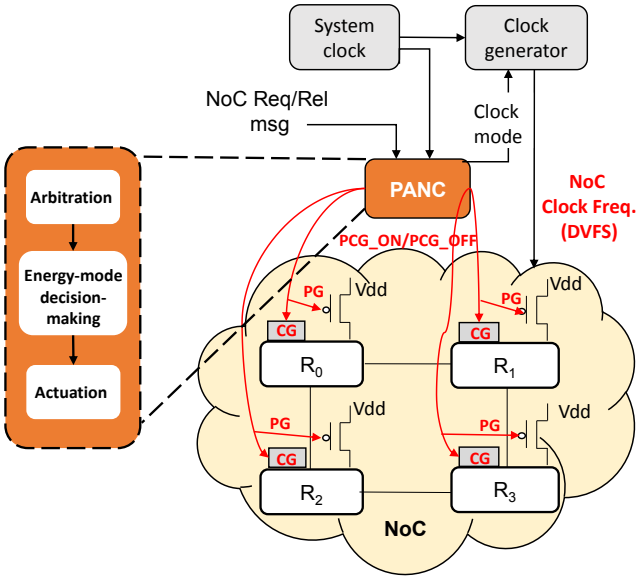

Fig. 2: NoC with the *PANC* approach, employing the integrated energy-savings schemes, *PG*, *CG*, and *DVFS*

In addition to that, *PANC* sends *PCG_OFF* messages (power/clock-gating off) to wake up all powered off routers and activate their clock before it acknowledges a sender. For more details about *PANC* functionality (in which *PANC* determines when to apply what energy-savings scheme), and scalability, the interested reader can consult the work of Kadeed et al. [5].

ACKNOWLEDGMENTS

This work has been funded by the German Research Foundation (DFG) within the project ER168/32-1.

REFERENCES

[1] B. K. Daya and et al., "SCORPIO: a 36-core research chip demonstrating snoopy coherence on a scalable mesh noc with in-network ordering," in *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3. IEEE Press, 2014, pp. 25–36.

[2] A. Kostrzewa, T. Kadeed, B. Nikolić, and R. Ernst, "Supporting dynamic voltage and frequency scaling in networks-on-chip for hard real-time systems," in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2018, pp. 125–135.

[3] T. Kadeed, E. A. Rambo, and R. Ernst, "Power and area evaluation of a fault-tolerant network-on-chip," in *System-on-Chip Conference (SOCC), 30th IEEE International*. IEEE, 2017, pp. 190–195.

[4] T. Kadeed, S. Tobuschat, A. Kostrzewa, and R. Ernst, "Safe and efficient power management of hard real-time networks-on-chip," *Integration, the VLSI Journal*, 2018.

[5] T. Kadeed, S. Tobuschat, and R. Ernst, "Integrated energy control for hard real-time networks-on-chip," in *2019 IEEE Real-Time Systems Symposium (RTSS)*, 2019.

A Gem5 Multi-OS Mixed-critical Many-core Simulation Model for Self-aware Systems

Eberle A. Rambo, Robin Hapka, Rolf Ernst
Institute of Computer and Network Engineering
TU Braunschweig
Braunschweig, Germany
{rambo,robinh,ernst}@ida.ing.tu-bs.de

***Abstract*—Mixed-critical systems are the natural evolution of safety-critical embedded systems. They can make more efficient use of the abundant computing power in modern architectures. Despite the strong theoretical work on mixed-criticality and well-developed analytical frameworks, simulation models have not been developed at the same pace. This work presents a mixed-critical, multiprocessing simulation environment with multi-OS support developed in Gem5. The highly configurable tile-based many-core architecture model with network-on-chip enables the exploration and evaluation of self-awareness techniques for mixed-criticality on an experimental hardware platform.**

***Index Terms*—simulation, mixed-criticality, many-core, self-awareness**

I. Introduction

The Information Processing Factory (IPF) project has recently introduced the abstraction of modern and future complex multiprocessing architectures as self-aware information processing factories: the IPF paradigm [1]–[3]. These factories consist of a set of highly configurable resources, such as processing cores and interconnects, whose use is monitored, planned, and configured during runtime. Continuing with the analogy, managing a factory requires multiple considerations, such as efficiency, availability, reliability, integrity, and timing. IPF conquers the complexity of managing such systems by hierarchically decomposing the challenges. These are then addressed by different mechanisms that co-exist in the factory.

To enable a flexible and fast experimentation and evaluation with hardware-software co-design, we have derived a full-system simulation model in Gem5 [4]. By composing new and existing models, we created a multiprocessing platform suitable for mixed-critical, self-aware systems. That requires support for different operating systems (OSs) and real-time operating systems (RTOSs) that co-exist in the platform, architectural support for mixed-criticality, and a high degree of configurability for self-awareness. This paper summarizes a mixed-critical many-core simulation model for self-awareness with multi-OS support.

II. The Simulation Model

A. Overview of the Architecture

The simulation model consists of a tile-based many-core architecture for mixed-critical real-time systems, as required by the IPF paradigm [3]. Fig. 1 illustrates an instance of the architecture, based on [5], which is composed of a number of tiles connected to each other by a network-on-chip (NoC). The platform size, i.e., the number of tiles and the number of routers in the network are parameterizable. A tile is connected to a NoC router by a network interface (NI), and up to four tiles can be connected to the same router. Common in academia, the regular 2D-Mesh is the default topology for the NoC. The NoC model also supports irregular, custom topologies, which are common in industry.

We acknowledge financial support from the DFG Grant ER168/32-1.

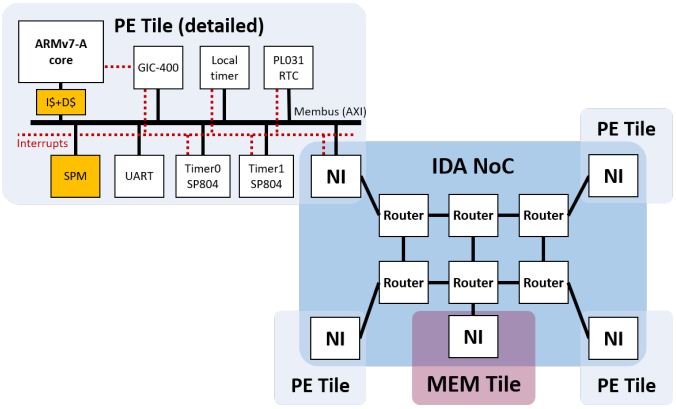

Fig. 1. Overview of the modeled tile-based architecture, with a detailed tile.

B. Tiles

Tiles can be complete systems containing one or more cores in a cluster, interrupt controller, various high and low-bandwidth peripherals, on-chip memories, off-chip memory interfaces, and I/O interfaces. We distinguish different types of tiles based on their purpose: processing tiles (PE tiles), memory tiles (MEM tiles), and I/O tiles (IO tiles). The first two tile types are depicted in Figure 1. Initially, the model has a number of homogeneous PE tiles and one MEM tile.

Each PE tile consists of an ARMv7 core, a local scratchpad memory (SPM), and peripherals connected to a tile-local AMBA bus. The local bus is connected to the NI, which acts as a gateway to the rest of the chip. The model supports multiple cores in a tile and the number of cores is currently limited only by components, such as the interrupt controller (ARM's GIC-400). The memory hierarchy can be customized as required: with or without caches and with or without a local scratchpad memory. The model with caches currently supports software-based coherence. Hardware-based coherence for mixed-critical real-time systems is an open problem, especially for NoC-based ones, and therefore potential future work.

The MEM tile contains a read-only memory, a DDR memory controller, and a tile-local bus connecting them to the NI.

Tiles can be easily customized, a characteristic inherited from Gem5 [4]. Peripherals, memory modules, and interfaces can be added and modified as required. That enables the exploration of mechanisms and architectural choices for self-awareness in mixed-criticality. The model can be extended to other Instruction Set Architectures (ISAs) depending on Gem5's support for full-system simulation with that ISA. Moreover, existing features in Gem5, such as dynamic voltage and frequency scaling (DVFS), are also supported.

All tiles are equipped with at least one NI, which is connected to the tile-local bus. The communication with and the access to resources and memories in other tiles is done through the NI.

C. Network-on-chip (NoC)

The IDA NoC provides the connectivity between tiles and resources in the system. Based on [5], the IDA NoC consists of routers, NIs, and links. The NoC implements wormhole switching, where variable-sized packets are composed of fixed-sized flow control units (flits); virtual-channel flow control, where flits transit through a number of virtual channels; priority-based arbitration; and deterministic source routing, where the route and virtual channel are defined in the NI.

1) Support for mixed-criticality: In order to make the performance of tiles sufficiently independent from each other, a requirement for mixed-criticality [3], [6], spatial isolation and bounded temporal interference are provided by the NoC. The spatial isolation is ensured by controlling the inter-tile communication and access to shared hardware resources on a whitelist basis: tiles can only communicate with other tiles and access resources if they are allowed by a system controller. That is implemented in the NoC routers, e.g., through priority-based arbitration with virtual-channel flow control, and in the NI through a memory protection unit with memory translation and an interrupt unit, both introduced next.

2) Memory management: The access to remote resources and memories is handled by the NI, which converts memory bus transactions into NoC packets and vice versa in the remote tile. Therefore, the NI features a memory protection unit with memory translation based on [5], which maps tile-local physical addresses to physical addresses on a remote tile. Additionally, mapped addresses also have an associated route and virtual channel that are used to configure the associated NoC packets. The configuration of the NIs can be done statically and modified dynamically by a trusted system controller.

3) Support for interrupt-based communication: The NI also features an interrupt interface that delivers interrupts to remote tiles' interrupt controllers. The configuration of this feature is analogous to the NI's memory unit, but instead of memory addresses, it maps local and remote interrupt numbers.

III. The Software

A. Multi-OS support

The simulation model supports the execution of a platform with multiple OSs. Currently, two RTOSs have been ported to the platform: FreeRTOS and μC-OS II. Each tile has an instance of an OS, resulting a multi-kernel setup. Future work includes porting and supporting non-real-time, best-effort OSs. Details on the use of multiple OSs in a mixed-critical, self-aware system is available in [6].

B. Inter-tile Communication

For inter-tile communication, a message passing interface (MPI)-compatible communication API has been created that allows the communication across tiles and different operating systems. It makes use of the interrupt and memory management features of the NoC for increased efficiency. Our MPI implementation currently supports the point-to-point communication subset of the MPI-3.1 standard.

C. Support for self-awareness

Self-awareness requires, among others, a high degree of configurability both in software and hardware. Self-awareness in a mixed-critical system must additionally handle constraints and requirements of the safety-critical workload while optimizing the execution of best-effort workload. The current model provides the functionality required by entities of a self-aware IPF system. For instance, by allowing the reconfiguration and repurposing of tiles, and the migration of tasks and tiles. The inter-tile communication infrastructure supports the reconfiguration of the system at runtime and keeps track of migrated, communicating tasks. It operates in conjunction with a system controller, responsible for the actual migration [3].

IV. Summary

This paper summarized a mixed-critical many-core simulation model for self-awareness with multi-OS support. The work was developed in the context of the Information Processing Factory (IPF) project, which currently researches self-awareness for mixed-criticality. The simulation model extends existing models in Gem5 while inheriting existing infrastructure and features, and enables the rapid experimentation and hardware/software co-design.

References

[1] N. Dutt *et al.*, "Conquering MPSoC complexity with principles of a self-aware information processing factory," in *Proceedings of the 11th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, ser. CODES'16, Pittsburgh, Pennsylvania, Oct 2016.

[2] A. Sadighi *et al.*, "Design methodologies for enabling self-awareness in autonomous systems," in *Proceedings of Design, Automation and Test in Europe Conference (DATE'18)*, March 2018.

[3] E. A. Rambo *et al.*, "The information processing factory: A paradigm for life cycle management of dependable systems," in *Proceedings of the 14th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, ser. CODES'19, New York, New York, Oct 2019.

[4] N. Binkert *et al.*, "The Gem5 Simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, Aug. 2011.

[5] B. Motruk *et al.*, "IDAMC: A many-core platform with run-time monitoring for mixed-criticality," in *2012 IEEE 14th International Symposium on High-Assurance Systems Engineering*. IEEE, 2012, pp. 24–31.

[6] E. A. Rambo *et al.* (2019) The information processing factory: Organization, terminology, and definitions.

Runtime Architect: Link Performance Design to Runtime Aspects

Adel Gasri
Link Software International
Tunis, Tunisia
adel.gasri@linkconet.com

Rafik Henia
Thales Research & Technology
Palaiseau, France
rafik.henia@thalesgroup.com

Laurent Rioux
Thales Research & Technology
Palaiseau, France
laurent.rioux@thalesgroup.com

Nicolas Sordon
Thales Research & Technology
Palaiseau, France
nicolas.sordon@thalesgroup.com

***Abstract*—Timing predictability is one of the most important concerns in real-time systems. Today, timing predictability is however challenged by the high complexity of modern real-time systems, which is the consequence of the significant increase of the number of implemented software functions and the required computing and communication resources. Model-driven technologies (e.g. design modelling and timing verification tools) are considered as best adapted for the timing prediction of complex real-time systems. However, there is a lack of solutions ensuring that the model reflects the real system content and is causally connected with it. In this context, we present a demonstration of the Runtime Architect tool suite that permits to system architects linking the system model with the system runtime aspects. This allows setting up a continuous performance engineering cycle between design and runtime, thus ensuring the quality of the running system while reducing the design and development efforts and costs. Runtime Architect encapsulates the Time4Sys real-time platform in order to ease the integration in any runtime, design or timing verification environment. A demonstration on an industrial avionics use-case will be presented to show the pertinence of our solution.**

Keywords—real-time verification, runtime, performance engineering, model-based design

I. INTRODUCTION

Electronic systems with real-time performance constraints are found in many different application domains such as space, automotive, railway, aerospace, medical monitoring and imaging, industrial process control, robotics, etc. A common trend amongst all these systems is the significant quantitative (number of functions) and qualitative (required performance) increase of the functions to be implemented in order to respond to the growing needs for connectivity, automation, autonomy, security, etc. Among the different activities related to the development of real-time systems, Performance Engineering (PE) is one of the most challenged by this trend: in order to face the qualitative and quantitative increase of functions, it is extremely important to make the best use of the available computing and communication resources while ensuring that the real-time constraints are met (in critical systems no risk can be taken on timing aspects since they represent a high safety concern, e.g. functions of the flight control computer must be executed in time to ensure aircraft flight stability; in non-critical systems timing delays may strongly impact the quality of service, e.g. image glitches in video surveillance). This ultimate objective calls for modern PE practices.

In PE, model-driven technologies such as tools for performance design and timing verification seem best adapted for this purpose and promise significant productivity gains, which have been proven valid in several studies (e.g. reuse of models in future developments, better understanding of the real-time behavior through an abstract view focusing on timing aspects, etc.). However, there remains a major problem aggravated by the growing systems complexity: Does the implemented system at runtime behave as designed? Without some form of consistency guarantee, the relationship between the system model and its implementation will be hypothetical, and the above mentioned gains brought by the model-driven technologies will be lost. As a consequence, one of the major challenges today in the PE of real-time systems is the integration of design models and runtime aspects. The timing behavior at runtime has to be matched with the design in order to identify the timing failures in design and deviations from the real-time requirements.

In this perspective, we have developed the Runtime Architect tool suite. It allows the architect performing a continuous system PE cycle between design and runtime, thus ensuring the quality of the running real-time system while reducing the design and development efforts and costs, and getting valuable feedback that can be used to boost the productivity and provide lessons-learnt for future generations of the product. In the demo session, we will present the tool suite Runtime Architect and will run a demonstration using a critical industrial real-time application from avionics: the flight management system.

II. RUNTIME ARCHITECT

Runtime Architect incapsulates the MARTE [1] based open source real-time platform Time4Sys [2]. By relying on the Time4Sys design model and the Time4Sys trace model, Runtime Architect automatically benefits from current but also future connections to Time4Sys of the various existing model-driven PE tools such as design tools (Capella [3], Papyrus [4], Matlab [5], etc.), scheduling analysis and simulation tools (Mast [6], Cheddar [7], SimEvents [8], etc.) as well as tracing tools (Trace Compass [9], LTTng [10], CTF [11], etc.) all at once. This guarantees high flexibility and add a valuable agnostic character to Runtime Architect since it is possible to easily integrate it in any runtime, any design and any scheduling verification environment.

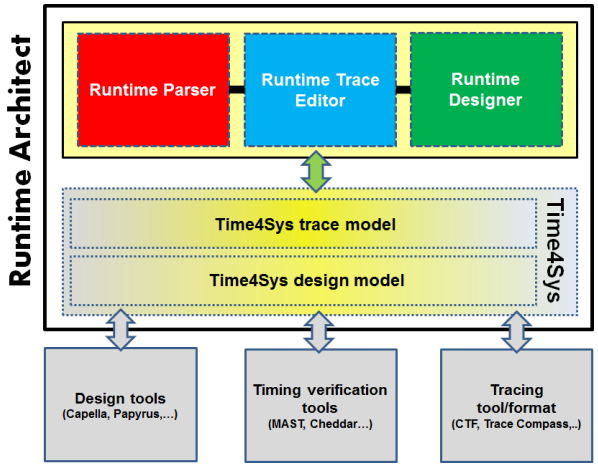

Fig. 1. Runtime Architect structure and interaction with design, scheduling verification and tracing tools

Runtime Architect exploits design models of real-time systems in Time4Sys, which typically include strong timing requirements, in combination with collected runtime traces in Time4sys originating from the system execution or communication to achieve a continuous system PE cycle between design and runtime. This is achieved through the interaction of three modules that complement each other: Runtime Parser, Runtime Trace Editor and Runtime Designer. The design and the trace models in Time4Sys are at the core of the modules. The structure of Runtime Architect, as well as its interaction with design modelling, runtime tracing and timing verification tools is illustrated in Figure 1. The combined use of all three modules in connection with Time4Sys allows **integrating feedbacks on the real-time behavior obtained through the processing of runtime traces directly into system design**, since it is the most suitable environment for architects to take decisions, thus increasing the confidence that the timing performances will meet the requirements.

A. Runtime Parser

It parses the runtime traces in Time4Sys to provide valuable feedbacks on system execution to the architect. This includes various numerical and graphical statistics for several timing properties as well as the identification of performance bottlenecks or timing errors encountered in the traces such as deadline misses or buffer overloads. It also ensures consistency between timing properties in runtime and design. This is done by computing a mathematical model of the runtime performance behavior of the system. This model is then confronted to the one included in the Time4Sys design model.

B. Runtime Trace Editor

It is at the same time a graphical trace viewer and editor for system execution/communication. It offers graphical representations of common components and concepts building together the real-time behavior of the system (processors, tasks, functions, communication channels, blocking times, pre-emptions, etc.). This allows monitoring runtime traces to understand the timing behavior and identify performance bottlenecks. It offers an interface to connect to any analysis/simulation tool to graphically represent the computed/simulated scheduling performance results. It also offers the capacity to edit the displayed runtime traces with the objective of exploring design alternatives based on the system runtime behavior before investing time and efforts in re-implementing and testing. E.g. it is possible to select any graphical component in the displayed trace and modify its timing characteristics such as increasing/decreasing the duration of a function execution, shifting the transmission of a frame on a bus or the activation of a task on a processor, etc.

C. Runtime Designer

It allows virtually searching for critical execution scenarios for system performance, correcting timing errors if any, and exploring design alternatives. For this purpose, Runtime Designer interacts with the Runtime Trace Editor as well as the Time4Sys system design model to interpret the edition of the runtime traces by the architect. Based on that interpretation, it re-computes the runtime trace, e.g. an increased execution duration of a given task instance decided by the architect in Runtime Trace Editor will make Runtime Designer rebuild the whole runtime trace taking in consideration the tasks priorities as defined in the Time4Sys design model. Another example: by shifting the activations of some tasks in Runtime Trace Editor, the architect may distribute the execution load of the tasks over the time, thus reducing their respective response times.

Once an explored correction or design alternative is validated by the architect, he can decide to integrate the modified timing properties in the Time4Sys design model. Note that the correction/exploration performed using Runtime Designer is specific to the observed timing behaviour in the runtime traces. It must be complemented with an overall system timing verification and optimization activity performed using any scheduling analysis or simulation tool connected to Time4Sys as illustrated in Figure 1.

III. DEMONSTRATION

We will present a demonstration using a use-case from the avionic domain: the Flight Management System (FMS). The FMS is a critical application embedded in the aircraft in charge of computing the trajectory along a flight plan during the flight, while minimizing fuel consumption and time to arrival. It term of complexity, it is representative of real industrial applications (various activation patterns, computing intensive parts, mixed criticality features, etc.).

REFERENCES

[1] Modeling and Analysis of Real-time and Embedded systems: https://www.omg.org/omgmarte/
[2] Time4Sys: https://www.polarsys.org/time4sys/
[3] Capella://www.polarsys.org/capella/
[4] Papyrus: https://www.eclipse.org/papyrus/
[5] Matlab: https://www.mathworks.com/products/matlab.html
[6] Modeling and Analysis Suite for Real-Time Applications: https://mast.unican.es/
[7] Cheddar: http://beru.univ-brest.fr/~singhoff/cheddar/
[8] SimEvents: https://www.mathworks.com/products/simevents.html
[9] Trace Compass: https://www.eclipse.org/tracecompass/
[10] Open source tracing framework for Linux: https://lttng.org/
[11] The common trace format: https://diamon.org/ctf

toki: A Build- and Test-Platform for Prototyping and Evaluating Operating System Concepts in Real-Time Environments

Oliver Horst
fortiss GmbH – Research Institute of the Free State of Bavaria
Guerickestr. 25, 80805 Munich, Germany
Email: horst@fortiss.org

Uwe Baumgarten
Technical University of Munich
Department of Informatics, Germany
Email: baumgaru@in.tum.de

***Abstract*—Typically, even low-level operating system concepts, such as resource sharing strategies and predictability measures, are evaluated with Linux on PC hardware. This leaves a large gap to real industrial applications. Hence, the direct transfer of the results might be difficult. As a solution, we present *toki*, a prototyping and evaluation platform based on FreeRTOS and several open-source libraries. toki comes with a unified build- and test-environment based on Yocto and QEMU, which makes it well suited for rapid prototyping. With its architecture chosen similar to production industrial systems, toki provides the ground work to implement early prototypes of real-time systems research results, up to technology readiness level 7, with little effort.**

I. Introduction

Currently, most applied real-time systems research prototypes are developed and evaluated on top of Linux on PC hardware. This leaves a large gap between real industrial applications in that field and the prototype. In case of low-level operating system (OS) concepts concerning, *e.g.*, context switch times, resource sharing, intra-node communication, and predictability, the drawn conclusions could even be void due to the completely different nature of the industrial platform. Furthermore, we see a lack in practical examinations of which latency and how much temporal predictability, in the sense of [1], is achievable with certain configurations (*e.g.*, software architectures and predictability measures). Hence, we see the need to ease constructing early prototypical implementations of research results on relevant hardware in relevant environments.

On the one hand, Linux seems to be a good choice here. It is available for a wide variety of hardware platforms, provides excellent third-party library support, and can fulfill real-time requirements to some extent. However, it is a complex task to configure Linux in a way that its influence on low-level performance benchmarks is negligible or at least predictable. Moreover, integrating own concepts into the Linux kernel requires detailed knowledge about the kernel sources and its concepts. Hence, even though specialized distributions, such as LITMUS^{RT} [2], can drastically reduce the configuration effort, the integration complexity remains as main issue.

Industrial-grade real-time operating systems (RTOSs), on the other hand, provide considerably less influence on performance measurements, but at the costs of usability. They are either delivered as minimal systems, like FreeRTOS [3], which lack tooling support and provide not much more than a scheduler, or as sophisticated platforms tailored to specific industrial fields, such as AUTOSAR [4], which come along with royalty fees, tooling, and complex, configurable software stacks.

As an exception, the GENODE OS framework [5] comes with toolchains for several hardware platforms and provides a convenient, production ready RTOS. Unfortunately, GENODE's overall orientation to safety and security severely hinders rapid prototyping. Its micro-kernel based design requires that all adaption and changes to the OS obey the strict isolation mantra, which is conceptually challenging and time consuming.

Therefore, we see the need for a minimal, yet flexible real-time system framework that provides comfort similar to commercial platforms, but comes without a complex software architecture and strict inherent design concepts. Ideally, the framework should be fully based on open-source software, provide an integrated standard C-library, a target toolchain, configuration tools, and an emulation environment for testing. A plus would be the possibility to safety certify the system.

Accordingly, we present *toki*, a flexible and configurable OS framework, close to production industrial systems, but without the hassle of complex software architectures.

II. The Build- and Test-Platform Toki

toki, Japanese for "when an action occurs", is a build- and test-environment constructed around FreeRTOS or SafeRTOS [6], respectively. toki's main goal is to compose an easy to use rapid prototyping platform to develop and evaluate new operating-system- and intra-node-communication-concepts for cyber-physical systems on commodity microcontrollers. Therefore, toki combines the following open-source projects into the unified architecture and build-system illustrated in Fig. 1: FreeRTOS [3] (v10.0.0), newlib [7] (v3.0.0), libXil / libXilPm [8] (v2019.1), and lwIP [9] (v2.0.3). These base components

This work was supported by the European Union (EU) under the Horizon 2020 program, projects TAPPS (Trusted Apps for open CPSs) and 5GCroCo (5G Cross Border Control), and the German Federal Ministry of Economics and Technology (BMWi) under the Smart Service World program, project PASS (Platform for Automotive Apps Guaranteeing Security and Safety).

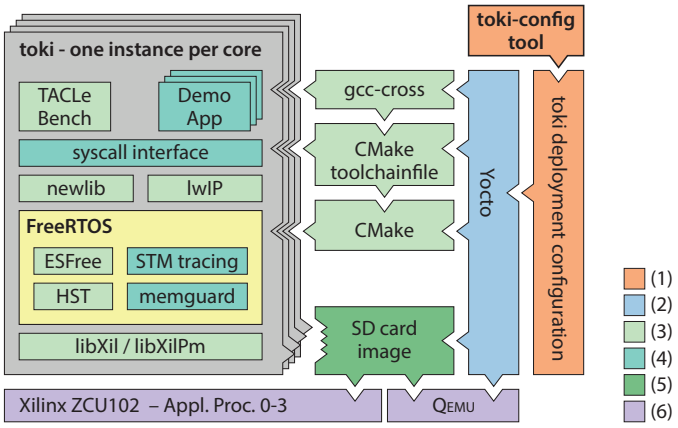

Figure 1. Sketch of toki's architecture and build-system, showing the included components (3,4), differentiated among integrated (3) and newly created (4) components, and their relations. Given a deployment configuration (1), Yocto (2) builds all included components and collates them into a SD card image (5), which can be deployed and tested on the hardware or in QEMU (6).

are supplemented by the ESFree Scheduling Library [10], HST User Mode Scheduler [11], and TACLe benchmarks [12].

These components were chosen under the following aspects: *(i)* the simplicity to modify and extend their code base, *(ii)* their ability to closely mimic the software stacks of comparable industrial systems, *(iii)* the flexibility of their design regarding rapid prototyping, and *(iv)* compatible software licenses.

At platform-level, the toki build-system (Fig. 1) utilizes Yocto [13] to provide a self-contained build-environment that builds all components of toki together with their dependencies (*e.g.*, a GCC cross-compiler). At project-level, we ensured proper include prefixes for all components, by restructuring their sources and adding a CMake [14] based build-system, where needed. This reorganization is conducted by a dedicated Python script, to allow nearly automatic upstream pulls.

To allow software developers to benefit from the code insight and debugging capabilities of their favorite IDE, single-core deployments of toki can be built by CMake. Production-ready images of multi-core deployments, on the other hand, can only be built by Yocto. toki deployment configurations, in general, are specified either manually or with assistance of the toki-config tool, and guide Yocto in the build process.

Besides the changes required for the build-environment, we contribute the following new features to FreeRTOS on ARMv8: asymmetric multiprocessing boot support, a newlib syscall interface, a memguard [15] implementation, and software-tracing support via ARM's system trace macrocell [16].

The toki build-environment is seamlessly integrated with a virtual test-environment. Hence, all images built by Yocto can either be tested on the real hardware or its emulated counterpart. The emulation is handled by a tailored QEMU [17], built by Yocto, which enables a fully virtual development cycle.

Currently, toki is solely tested and ready-to-run on the Xilinx Zynq UltraScale+ MPSoC platform [18], a contemporary ARMv8 multi-core SoC with an integrated FPGA. This platform was selected, because of *(i)* Xilinx's extensive software support for it, including a bare-metal driver kit, and *(ii)* the included FPGA, which allows us to increase the accuracy of our performance evaluations. Nevertheless, toki was designed with portability in mind; hence, it can easily be ported to other target platforms, *e.g.*, the STM32Cube by STmicro [19].

In the future, we plan to conduct measurements to compare the interrupt handling latency of toki with other software stacks and extend toki with FreeRTOS+POSIX [20], a communication middleware, and precision time protocol (PTP) support.

Feel free to try out the latest version of toki, by downloading it from: https://git.fortiss.org/toki

III. Demonstration

We will demonstrate toki through two showcases:

1) A video of the TAPPS project's [21] final demonstrator, showing the on-the-fly installation of an application into the real-time critical control path of the throttle control of a production electric motorcycle, realized with toki.
2) A live-demo of toki's configuration, build, simulation, and deployment cycle on the example of memory benchmarks deployed to distinct cores and regulated by memguard.

The first showcase focuses on the real-time capabilities and applicability to industrial use-cases, and the second on the configuration flexibility and measurement capabilities of toki.

Acknowledgment

We would like to thank all contributors to toki for their work, namely: Martin Jobst, Johannes Wiesböck, Dorel Coman, Ulrich Huber, Mahmoud Rushdi, Tuan Tu Tran, Firas Trimech, Raphael Wild, Andreas Ruhland, and Dhiraj Gulati.

References

[1] B. Sun *et al.*, "Definitions of predictability for cyber physical systems," *J. of Syst. Architecture - Embedded Syst. Des.*, vol. 63, pp. 48–60, 2016.
[2] J. M. Calandrino *et al.*, "LITMUSˆRT : A Testbed for Empirically Comparing Real-Time Multiprocessor Schedulers," in *27th IEEE Int. Real-Time Syst. Symp. (RTSS)*, Dec. 2006, pp. 111–126.
[3] https://www.freertos.org.
[4] AUTOSAR, "Classic platform release 4.3.1," AUTOSAR Std., Dec. 2017.
[5] N. Feske, *GENODE Foundations – Operating System Framework 19.05*. GENODE Labs, 2019.
[6] https://www.highintegritysystems.com/safertos/.
[7] https://sourceware.org/newlib/.
[8] https://github.com/Xilinx/embeddedsw.
[9] https://savannah.nongnu.org/projects/lwip/.
[10] R. Kase, "Efficient Scheduling Library for FreeRTOS," Master's thesis, KTH Information and Communication Technology, 2016.
[11] F. E. Páez *et al.*, "FreeRTOS user mode scheduler for mixed critical systems," in *6th Argentine Conf. on Embedded Syst. (CASE)*, Aug. 2015.
[12] H. Falk *et al.*, "TACLeBench: A Benchmark Collection to Support Worst-Case Execution Time Research," in *16th Int. Workshop on Worst-Case Execution Time Analysis (WCET)*, 2016, pp. 2:1–2:10.
[13] https://www.yoctoproject.org.
[14] https://cmake.org.
[15] H. Yun *et al.*, "MemGuard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms," in *19th IEEE Real-Time and Embedded Technol. and Appl. Symp. (RTAS)*, Apr. 2013.
[16] *CoreSight System Trace Macrocell – Technical Reference Manual*, r0p1 ed., ARM Limited, Dec. 2010.
[17] https://www.qemu.org.
[18] https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html.
[19] https://www.st.com/content/st_com/en/stm32cube-ecosystem.html.
[20] https://www.freertos.org/FreeRTOS-Plus/FreeRTOS_Plus_POSIX/.
[21] http://www.tapps-project.eu.

Modelling and Timing Analysis of Real-time Applications on Evolving Automotive E/E Architectures using Rubus-ICE

Alessio Bucaioni*, John Lundbäck‡, Mattias Gålnander‡, Kurt-Lennart Lundbäck‡,
Mohammad Ashjaei*, Matthias Becker†, Saad Mubeen*
* Mälardalen University, Västerås, Sweden
‡ Arcticus Systems, Järfälla, Sweden
† KTH Royal Instituite of Technology, Stockholm, Sweden
*{alessio.bucaioni, mohammad.ashjaei, saad.mubeen}@mdh.se, †mabecker@kth.se
‡{john.lundback, mattias.galnander, kurt.lundback}@arcticus-systems.com

***Abstract*—The automotive E/E architectures are evolving from the traditional distributed architectures to upcoming consolidated domain architectures and possibly future centralised architectures. This paper demonstrates modelling and timing analysis of real-time embedded systems on contemporary automotive E/E architectures using the Rubus-ICE tool suite. The Rubus concept and tool suite, developed and evolved based on close academic-industrial collaboration, have been used in the automotive industry for over 25 years. The paper also demonstrates recent extensions and discusses proposals to support the modelling and timing analysis of the systems on future E/E architectures.**

I. Introduction

Automotive software has been evolving and growing in complexity at a staggering pace for the past couple of decades [1]. The advanced features in contemporary and upcoming automotive software systems, e.g., Advanced Driver Assistance Systems (ADAS), require high levels of computational power and high data-rate low-latency on-board communication that is well beyond the capacity of traditional Electronic Control Units (ECUs) and on-board networks respectively. Consequently, the traditional distributed Electrical/Electronic (E/E) architectures began to pave way for the upcoming advanced consolidated domain and centralised architectures [2], [3]. The progression and evolution of the automotive E/E architectures is depicted in Fig. 1.

The advanced automotive E/E architectures are envisioned to leverage powerful ECUs in the form of heterogeneous computing platforms (e.g., containing CPUs, GPUs and FPGAs) that are connected by high-bandwidth and low-latency on-board backbone networks [1]. However, model-based development [4], [5] of predictable real-time embedded software on these platforms opens up several new challenges, including the modelling of embedded software enriched with timing properties, extraction of timing models from the software architectures, supporting end-to-end timing analysis of the software architectures, automatically generating code from the timing verified software architectures, and providing a predictable run-time environment.

This demo will present the model-based software development process (mainly modelling, timing analysis and synthesis) for real-time embedded systems on distributed automotive E/E architectures using the Rubus-ICE tool suite[1]. Rubus-ICE has been used in the automotive industry for over 25 years by several OEMs and Tier-1 companies (e.g., Volvo, BAE Systems, Hoerbiger, Knorr Bremse, BorgWarner, among others) for model-based development of predictable real-time embedded systems. The demo will also showcase recent extensions in Rubus-ICE to support the modelling of these systems on domain E/E architectures. In addition, the demo will present the ongoing challenges concerning the support for modelling and timing analysis of these systems on the upcoming and future centralised automotive E/E architectures.

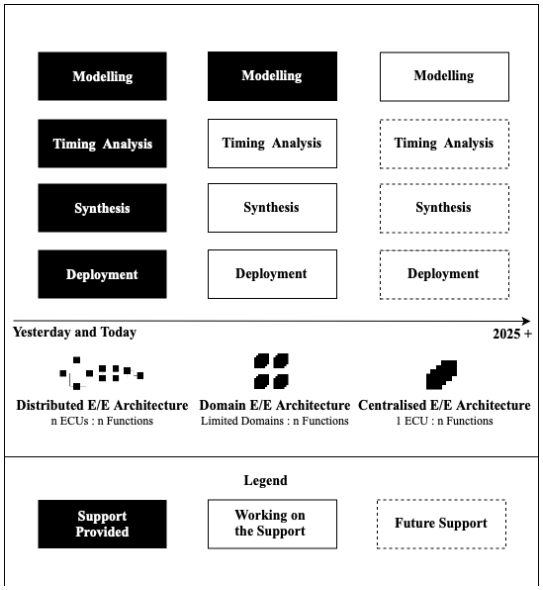

Fig. 1. Support for model-based development of real-time embedded systems in Rubus-ICE with the progression of automotive E/E architectures.

II. Modelling and Timing Analysis with Rubus-ICE

A. Distributed E/E Architectures

Rubus-ICE provides a full-fledged model-based development support for real-time systems on distributed E/E architectures. The development support includes modelling of software architectures and timing information, end-to-end timing analysis of the software architectures, automatic generation of timing verified code from the software architectures, deployment and execution on predictable run-time environment. Fig. 2

[1]Rubus-ICE Integrated component model Development Environment, http://www.arcticus-systems.com

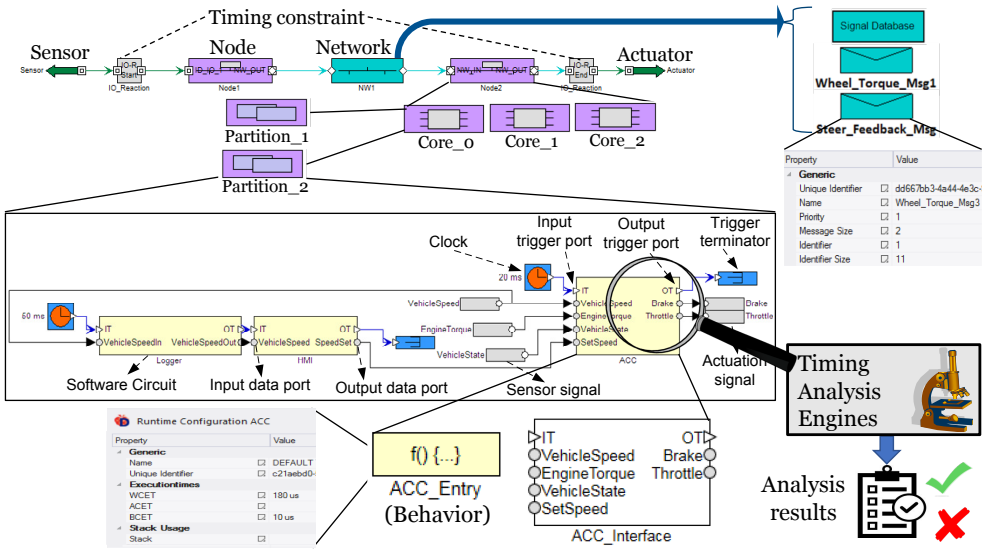

Fig. 2. Example of software development for real-time embedded systems on various Automotive E/E architectures using Rubus-ICE.

shows a screenshot of an example real-time system modelled and analysed on a distributed automotive E/E architecture.

B. Domain E/E Architecture

Domain E/E architectures employ more powerful processing units for replacing the constellation of single-core ECUs employed in contemporary distributed E/E architectures. Rubus-ICE currently supports modelling of software architectures and expressing timing information on the software architectures of real-time systems that are deployed on these architectures. For example, Fig. 2 shows the model of a software architecture of a real-time system that is deployed on a tri-core node with multiple partitions per core. The support for timing analysis and predictable run-time support for real-time systems on these architectures is an ongoing work.

C. Centralised E/E Architectures

The shift towards centralised E/E architectures requires integration of heterogeneous software (with respect to workloads, activation semantics, data-flow semantics, real-time requirements and safety requirements [6]) on heterogeneous hardware comprising of certified traditional processors and general-purpose high-performance processors with accelerators. Rubus-ICE currently supports the specification of heterogeneous software with various types of real-time properties and requirements, safety requirements and criticality levels (different Automotive Safety Integrity Levels (ASILs) A to D according to the ISO 26262 functional safety standard for road vehicles), activation semantics (time triggered, event triggered), data-flow semantics (synchronous, independent activation, task chains) and workloads. This demo will also present and discuss some of our recent proposals for enriching Rubus-ICE with fine-grained modelling elements that allow to model heterogeneous computing platforms, e.g., the new modelling elements that are marked in red in Fig. 3.

III. Summary

Automotive E/E architectures are steadily evolving. With that, the corresponding design flow is evolving alongside. This paper demonstrates modelling and timing analysis of real-time embedded systems on contemporary automotive E/E architectures using the Rubus-ICE tool suite. In addition, recent advancements that address challenges of future architectures are demonstrated.

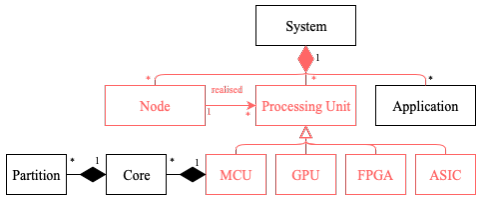

Fig. 3. Proposed new elements for modelling and specifying timing information on heterogeneous computing platforms.

Acknowledgment: This work is supported by the Swedish Knowledge Foundation (KKS) through the projects A-CPS and HERO, and by the Swedish Governmental Agency for Innovation Systems (VINNOVA) through the projects PANORAMA and DESTINE. The authors thank the industrial partners Arcticus Systems and Volvo Group for their valuable inputs.

References

[1] L. Lo Bello, R. Mariani, S. Mubeen, and S. Saponara, "Recent advances and trends in on-board embedded and networked automotive systems," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 2, 2019.

[2] Roland Berger, Consolidation in Vehicle Electronic Architectures, In Think: Aact, Jul., 2015. Available at: https://www.rolandberger.com/en/Publications/pub_consolidation_in_vehicle_electronic_architectures.html.

[3] H. Zinner, J. Brand, D. Hopf, Automotive E/E Architecture evolution and the impact on the network, IEEE802 Plenary, 802.1 TSN, Mar. 2019, Continental AG, available at: http://ieee802.org/1/files/public/docs2019/dg-zinner-automotive-architecture-evolution-0319-v02.pdf.

[4] G. T. Heineman and W. T. Councill, "Component-based software engineering: putting the pieces together," *Component-based software engineering: putting the pieces together*, pp. 33–48, 2001.

[5] S. Sendall and W. Kozaczynski, "Model transformation: The heart and soul of model-driven software development," *Software, IEEE*, vol. 20, no. 5, pp. 42–45, 2003.

[6] S. Saidi, S. Steinhorst, A. Hamann, D. Ziegenbein, and M. Wolf, "Future automotive systems design: research challenges and opportunities: special session," in *Proceedings of the IEEE International Conference on Hardware/Software Codesign and System Synthesis*, 2018, p. 2.