MeSViz: Visualizing Scenario-based Meta-Schedules for Adaptive Time-Triggered Systems

Babak Sorkhpour
Department of Electrical Engineering and Computer Science, University of Siegen
Siegen, Germany
Babak.Sorkhpour@uni-siegen.de

Roman Obermaisser
Department of Electrical Engineering and Computer Science, University of Siegen
Siegen, Germany
roman.obermaisser@uni-siegen.de

Abstract- A time-triggered Multi-Processor System-on-a-Chip (MPSoC) satisfies essential requirements of safety-critical embedded systems such as determinism and fault containment. Scenario-based meta-scheduling improves the flexibility in such a system by pre-computing different schedules at design time and allowing mode changes between schedules at run-time. Therefore, scheduling algorithms are required for solving the scenario-based temporal/spatial mapping of jobs to cores and messages to the Network-on-a-Chip (NoC). The meta-scheduling is highly relevant for automotive electronics. Safety-relevant automotive functions are based on time-triggered networks and can benefit from the meta-scheduling for improved energy-efficiency, flexibility, and dependability. To understand the complexity of developing these algorithms; developers need a global, abstract graphical visualization of their schedules. Engineers also need effective debugging and validation methods for scheduling algorithms to trace and view their output, thus saving time and cost. In this paper, we introduce MeSViz a tool dedicated to visualizing scenario-based meta-schedules as GANTT charts and tree-based graph.

I. Introduction

Many embedded systems are based on Time-Triggered (TT) networks and used in safety-critical applications (e.g., healthcare, space, military, nuclear stations, and aircraft). Efficient scheduling algorithms are required for such systems (e.g., mathematical programming, artificial intelligence, scheduling heuristics, neighborhood search [1]), where a failure would lead to severe consequences [2]. Scheduling limited resources among requesting entities is one of the most challenging problems in computer science [3]. In many articles, scheduling algorithms are explained by showing abstract tables of the essential information and data (e.g., tasks, messages, deadlines, times). However, it is hardly possible for humans to get an understanding and view of the entire schedules by only looking at the log files [3].

A visualization of a schedule enables the engineers to do sanity checks easily, e.g., checking and tracing the jobs, messages, nodes, and in scenario-based systems, the behavior for each scenario after an event. Although scheduling is an important issue in computer science, only a few visualization tools exist which help scientists to extend scheduling algorithms and models [3][4]. In scenario-based scheduling systems, the meta-scheduler (MeS [25]) generates specific schedules for each situation triggered by relevant events. To evaluate schedules, engineers need to compare, understand and debug schedules and this is a challenging matter in the MeS era. The output of most schedulers is in text format. It is a hard way to find problems especially when a huge number of schedules are generated and needed to debug or compare. This challenge is intangible when using text logs with big data or abstract graphics that cause wasting the engineer's mental resources. The majority of the schedule visualizers are designed to illustrate a single schedule and cannot cover multiple schedules in one scope. However, the generation of schedules via scenario-based scheduling solutions and algorithms for real-time multiprocessor systems is gaining increasing importance [6]. The approach of MeS is to add dynamic actions by computing several valid schedules, which will be dynamically chosen based on the system status [7]. Previous visualization tools show abstract schedules as graphical output in which they do not contain detailed explanations of the events and the schedules, differences or changes [2].

This paper introduces a scenario-based tool named the Meta-Schedule Visualizer (MeSViz). It can visualize single-schedules, meta-schedules and generate graphical data trees of schedules and their elements (e.g., nodes, IDs, events). MeSViz is designed to help developers to evaluate scheduling algorithms, models, and methods for adaptive TT systems. MeSViz is a tool to cover the gap of visualization in MeS for adaptive TT systems with scenario-based and event-based behaviors. This tool can display event locations, detail schedule changes and dependencies due to events. It focuses on visualizing schedules at three layers; the first layer presents individual schedules for each scenario, the second and third layers display multiple schedules for multi-scenario events. The rest of the paper is organized as follows:

The basic concepts, related work, and meta-scheduling are described in section II. In section III the visualization of schedules is described. The implementation

ISBN 978-3-8007-4524-1

© VDE VERLAG GMBH · Berlin · Offenbach

of the algorithms for the tool MeSViz and its features are explained in section IV. The case study and experimental results are given in section V. Finally; section VI concludes the paper with future research plans.

II. Basic Concepts and Related Work

A. Time-Triggered Systems

In time-triggered architectures, activities are triggered by the progression of global time. In such systems, time-triggered messages must be sent over the network at predefined instants and take precedence over all other message types [8][9]. For example, FlexRay [22] protocol is using TT pattern which Time is divided into communication cycles [15] and other in-vehicle network protocols including Local Interconnect Network (LIN), Media Oriented System Transport (MOST), Controller Area Network (CAN) [21] [16]. Time-triggered systems require development tools for computing offline schedules with temporal and spatial resource allocations that satisfy the application requirements such as precedence constraints and timing constraints. At runtime, dispatchers perform all activities controlled by these schedules.

B. Meta-Scheduling

Many research articles have been written about how to schedule a set of jobs in a system with limited resources [9]. However, few articles are focused on scenario-based MeS [7]. A real-time system has to execute concurrent tasks in such a way that all time-critical tasks meet their specified deadlines [10] . Every task needs computational and communication resources to proceed, and the scheduling problem is concerned with the allocation of all resources to satisfy all timing requirements [9]. For example, some techniques for message scheduling of FlexRay do not generate reliable or flexible schedules which they do not prepare any efficiency obligations in the fault event era [17]. In MeS to design easy, fast and accurate scheduling data structure, four models are using and defined:

1. Platform Model (PM): The first model is a physical model of the platform describing the on-chip resources including the cores, the switches, and their connectivity via the NoC.
2. Application Model (AM): The second model is the application model that defines jobs and their dependencies based on exchanged messages and their timing (e.g., worst-case execution times (WCETs) and deadlines).
3. Context Model (CM): The third model is the context model, in which system designers define all events and faults with implications on the application and the platform (e.g., jobs, switches, cores). These elements will be used as input conditions in the MeS to generate new schedules. This model is an important part of MeS which needs to control safety and helps facilitates fault-isolation and complexity management in smart and future vehicles generations [23] (e.g., car, flight controls, flying multicopters, self-flying taxi [24]).
4. Schedule Model (SM): The Forth model is the scheduling model, in which MeS store all generated schedule information (e.g., jobs run time and allocation, messages injection time and path) on a tree that each node is containing an SM. These elements will be used as input data in the MeSViz to visualization.

C. Meta-Scheduler

MeS [25] works with events as an essential prerequisite. For each relevant event, MeS creates a new schedule with edges and nodes of a graph tree. MeS is designed to generate schedules for anticipated changes of scenarios. When an event such as a fault (e.g., failed link or switch) or slack happens, the system can react by switching to precomputed schedules for these events [3]. In static or pre-run-time scheduling, a feasible schedule of a set of tasks is calculated offline [9].

D. Visualization of Schedules

An overview of existing tools and a short view of related work is shown in **Table 1**. These tools and their properties demonstrate that most of them are focused on simulation for generating the visualization [2]. Some tools such as Ghost, FORTISSIMO, ARTIST, and RTSSim, do not support either shared resources or multiple cores, which both are necessary features in adaptive TT multi-core systems [2]. RTSIM supports these features but focuses mainly on distributed systems [2]. MeSVis is designed for NoCs, multi-route messages according to TT control and real-time requirements.

Project	simulation	visualization	"live" simulation	shared resources	multiple cores	sporadic tasks	open-source	programming language
Alea2	—[1]	✓	✓	—	✓	✓	✓	Java
ARTISST	✓	✓	—	—	—	✓	✓	C++
Cheddar	✓	✓	—	✓	✓	✓	✓	Ada
FORTISSIMO	✓	—	—	—	✓	✓	✓	C++
gltraceviz	—	✓	—	✓	✓	✓	✓	C++
GHOST	✓	✓	—	—	—	✓	—	C
Jedule	—	✓	—	—	✓	✓	✓	Java
MAST	✓	—	—	✓	✓	✓	✓	Ada
Pajé	—[1]	✓	—	✓	✓	✓	✓	Objective C
Realtss	✓	✓[2]	—	✓	—	—	✓	TCL
RTsim	✓	✓	—	✓	✓	—[3]	—[4]	*Unknown*[4]
RTSIM	✓	✓	—	✓	✓	✓	✓	C++
RTSSim	✓	✓[2]	—	✓	—	✓	—	C
Schesim	✓	✓[2]	—	✓	✓	✓	✓	Ruby
STORM	✓	✓	—	✓	✓	✓	✓	Java
STRESS	✓	✓	—	✓	✓	✓	—	C
ViTE	—	✓	—	✓	✓	✓	✓	C++
VizzScheduler	✓	✓	✓	—	✓	—	—	Java

Table 1. Overview of existing real-time scheduling tools [2]

E. Requirements for Visualization

A visualizer is needed to show the contents of MeS including AM, PM, CM, and SM. To visualize multiple schedules, the visualizer needs to access the data tree of schedules to calculate and compare the difference of schedules. A basic schedule visualizer is capable of

displaying the following items:
1. Present job and message location, timing, and allocated resources.
2. Message routes and dependencies between the sender (job) and receiver(s).
3. Display textual description of message properties (e.g., duration, location).
4. Present textual description of job properties (e.g., runtime, WCET).
5. Current schedule node ID.
6. Flexibility to generate text and image output formats.

A meta-visualizer to visualize multi-schedules, at first needs to access the basic data which is generated via the basic visualizer and then use the CM contents to combine, analyze and generate extra details to display following items:
1. Display parent and child schedule nodes in one scope.
2. Report detailed event information in textual and graphical format.
3. Calculate and report the difference between parent and child nodes.
4. Display job and message location and duration changes in textual and graphical forms.
5. Display messages are routing changes.
6. Show total and removed schedules (e.g., invalid or pruned nodes)

F. Gantt mapping

For presenting schedules, a tool is needed that helps to understand schedules in one clear picture easily. The gantt chart is a straightforward solution to visualize the schedules. It is a better and easier solution than a textual format to understand schedules and their dependencies. This method is a common solution for human-resource scheduling, and in this paper, it is calling gantt mapping (e.g., **Figure 1**).

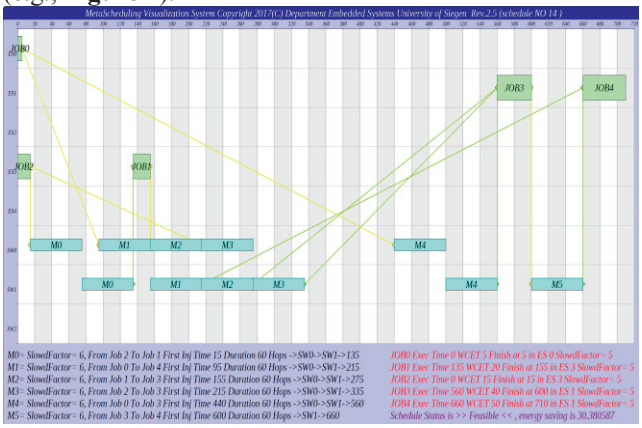

Figure 1. Schedule gantt map

G. Graph mapping

In MeS, the description of the events and the schedules in hierarchical graphs is necessary, and it is useful to handle and debug large and complex schedules. It needs a hierarchical description of events and resulting schedule changes. To define the schedules and event parameters, the graph model needs to be characterized. These characteristics are attributes of the nodes and edges of the graphs [5] (e.g., node ID, event). A graph mapping represents the event dependency between schedules [12] as shown in Figure 2.

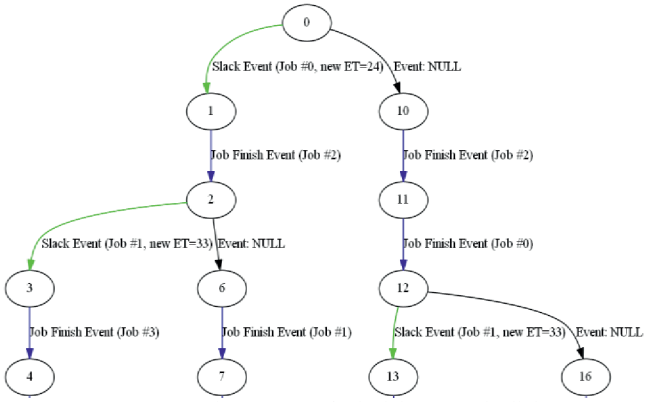

Figure 2. GraphViz output of schedule tree with all data

H. Visualization of schedule changes

When an event happens, it means that the execution scenario and system states are changed. It needs a new plan, and the SM has to be generated. However, it is important that the designer can comparable schedules to find out what changes happened. Mes generates this information, but a quick overview of the SM is needed which is called meta-visualization. MesViz has to be able to present when the event happens, what kind of changes were caused and which elements were changed. Examples are job and messages durations or allocation decisions as shown in **Figure 3**.

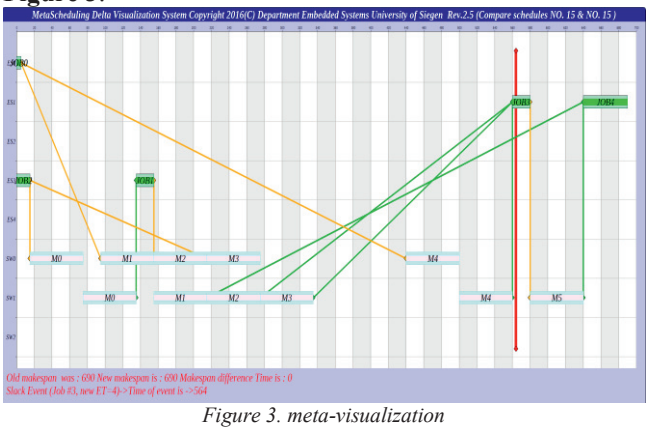

Figure 3. meta-visualization

III. IMPLEMENTATION OF VISUALIZATION

MeSViz uses ChartDirector to design extensive chart types. It utilizes a layered architecture, supporting graphical images formats (e.g., PDF, SVG, PNG, JPG, GIF and BMP) in C++ [11]. MeSViz has direct access to MeS classes for generating the gantt mapping using the ChartDirector library. It uses the GraphViz format for the GVEdit application and other graph generator tools. It uses multi-threading to increase performance.
After a schedule was generated, the visualizer is called and

ISBN 978-3-8007-4524-1

© VDE VERLAG GMBH · Berlin · Offenbach

started to read, and process AM, PM and SM directly from the scheduler class to provide input data to the Basic-Visualizer (BV). BV is the core of the meta-visualizer. To create meta-visual data, it needs more data of valid schedules which are included in the data tree. They contain specific AM, SM, CM, their top and down node addresses and IDs for parent and child nodes.

E. Outputs and formats

In MesViz, we have three graphical output file types and two text files.

1. Schd_Vis_XXXXX.png: This file is the basic output of each generated schedule which is visualized. XXXXX is the schedule ID numbering.

2. MetaSchd_Vis_XX_YY.png: This file is advanced output which visualizes, calculates and presents a difference of two related schedules (before and after an event) in the tree of the scheduled events (e.g., Figure 2). XX and YY are the IDs of relevant schedules and will be accessible and addressable in the output of GVEdit, and output text files. To generate this output, MesViz needs to read SM and AM, PM of parent and child nodes and events of each schedule and their IDs.

3. Output.txt: Schedule trees generate this file via a write_to_file class. This class is responsible for generating input formats of BV and meta-visualizer and text files of node connections and attributes which can be processed by GVEdit (e.g., **Figure 4**). This file includes all possibilities of nodes and events and shows all wrong or correct data which is built by the scheduler.

```
digraph G {
        0-> 1 [label= "Slack Event (Job #6, new ET=10)"color= green] ;
        1-> 2 [label= "Slack Event (Job #8, new ET=10)"color= green] ;
        2-> 3 [label= "Slack Event (Job #0, new ET=25)"color= green] ;}
```

Figure 4. Example a simply generated data with three node

4. Output2.txt: After removing duplicated nodes or null events, this file is generated to be used by GVEdit.

5. Graph_Tree.png: The last step is generating a graph mapping of meta schedules that GVEdit reads from output.txt, output2.txt and generate them. **Figure 2** is the result of the above example.

F. Composite tasks, time alignment, and node

Decision variables, constants, and constraints lead to a visual picture with all information aligned and allocated at accurate locations with semantic connections between elements via colored lines and text.

G. Single Schedule Gantt mapping

Each basic gantt map contains various information that presents all data of the schedule in one picture (cf. **Figure 1**). The description of these data contains:

1. Schedule ID on the top row.
2. Total makespan time in the top of a table column.
3. Node ID and switch ID at the left of a table row.
4. Message ID, sender, and recover job, injection time, duration, visited hops and switches on the bottom row with black color.
5. Job ID, execution time, starting time, WCET, finishing time and related nodes on the bottom column with red color.
6. Each job aligned with the exact start and finish time and node coordination with green color and its label.
7. Each message aligned at the exact time and switch coordination with blue color and corresponding labels.
8. Each sent message connected by a yellow line to its sender job and each received message linked by a green line to the receiver job.

H. Multi-Schedule Gantt mapping

For each existing event, a multi gantt map is created by the combination of two schedules for better detection of the differences between them. This information displays various changes which help to compare related schedules and provide a better understanding of the system activities (e.g., **Figure 3**). The description of these data contains:

1. Start event time with the vertical red line.
2. Parent schedule elements with a darker color than a child.
3. Display makespan changes.
4. Display event information.
5. Track which jobs and messages changed.
6. Parent and child IDs on top.

I. Graph mapping of meta-schedules

Graph mapping of meta-schedules is the last step of visualization. This phase is related to the size and complexity of models needed to use different graph visualizer tools. For simple models and in this paper Graphviz used. For big data, the yEd graph editor and Gephi can be used. Each graph contains and shows this information:

1. The schedule ID on each node.
2. Event details (e.g., the name of the event, job ID, execution time.)
3. Each event edge connection presented with a specific color (e.g., slack with light green or job finish with blue color).

IV. Example / Case Study

To evaluate basic schedules and meta-schedules visualized by MeSViz, we prepared input elements for MeS. It was run on a virtual cluster machine with 12 cores of an Intel Xeon E5-2450 2.2 GHz and 60GB RAM on Ubuntu 14.04.5 (GNU/Linux 3.13.0-93-generic x86_64).

To find out the behavior of MeSViz, it was run with one simple and a complex model.

B. Simple Model

For the first test, a simple model with limited conditions and elements for fast debugging was designed. The properties are as follows:

1. SM Content: The AM includes 5 jobs and 7 messages,

while the PM includes 7 nodes (5 cores and 2 switches). The CM includes 5 slack events with NewExecutionTime=50, 5 battery events with new energy levels, and 7 faults via node crashes.

2. *Output results:* The basic visualizer generates 94 gantt maps from the basic-scheduler (Figure 1) and MeSViz generates six gantt maps (Figure 3) from MeS. GVEdit generates the graph mapping with 37 nodes.

C. Complex Model

After the simple model study, a complex model was introduced to test MeS and MESViz better.

1. *SM Content:* The AM included 15 jobs and 15 messages, while the PM includes 20 nodes (16 cores and 4 switches). The CM included 14 slack events with NewExecutionTime, 5 battery events with new energy levels, 13 faults with node crashes, 14 faults with link faults and babbling idiots, and 14 faults with message omission events.

2. *Output results:* BV generates 3048 gantt maps from the basic-scheduler, and the meta visualizer generates 35 meta-visual gantt maps from MeS. GVEdit generates a graph map with 37 nodes.

D. Discussion

MeSViZ is capable of generating basic and meta-gantt maps and graph maps. Analyzing these results helps to find out several design faults:

1. *Incorrect configuration:* MeSViZ helps an to detect incorrect input data (e.g., a message does not connect to a job or makes a wrong loop between cores and switches). On another hand, if the MeS tool includes faulty constraints or conditions, then incorrect schedules will be generated. Finding these types of errors in text and log reports is a time-consuming and tedious task.For example, a design fault can be found in a complex schedule in **Figure 5**, where job14 is the sender of M14 and job11 is the receiver of M14.

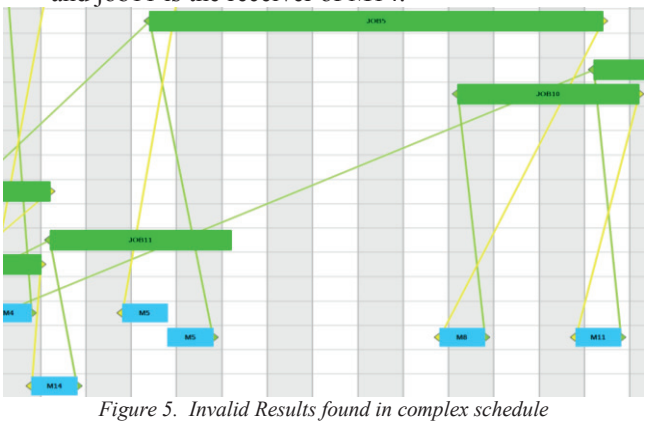

Figure 5. Invalid Results found in complex schedule

The problem is the execution time of job11, which finishes earlier than M14. In another example, job5 finished later than M8 and job10 started earlier than M8, which means that the crossed lines for messages indicate a wrong schedule.

2. *Mes Incorrect data:* When the visualizer builds invalid results form MeS data then we will have wrong results in MeSViz as shown for example in **Figure 6**. M12 in the old schedule has a right connection to the sender and receiver jobs (12-13), but in the new schedule, it has a crossed line.

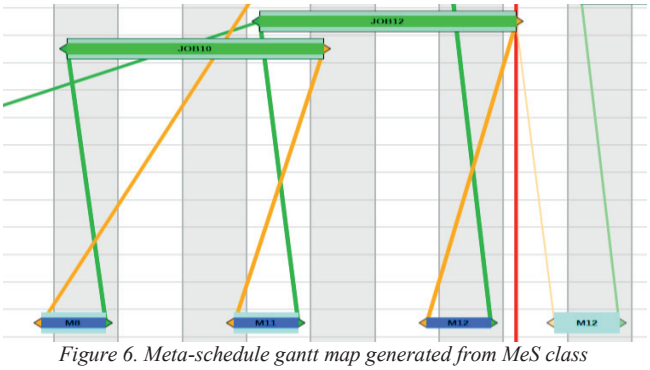

Figure 6. Meta-schedule gantt map generated from MeS class

3. *Incorrect pointer references:* Other problems that we found were wrong references in the MeS tree for schedule nodes. It is caused when we do not care for the pruning of zero references in the MeS data tree (e.g., Figure 7). Other problems are shown on the left side of Figure 7.

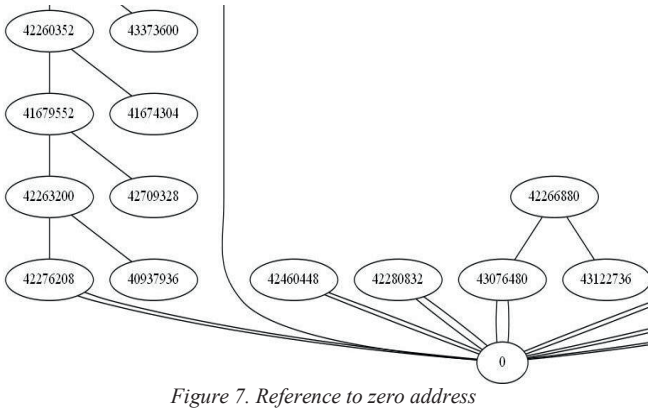

Figure 7. Reference to zero address

4. *Event trace:* The difference of edge colors in graph map helped better-finding event changes and behaviors (e.g., Figure 2).

5. As a significant visualization result, too fast evaluation schedules needed to attention direction and slope of the connection lines the sender and receiver job to each message.
   In Figure 5 and Figure 6 system designer easily can find which job or message is not aligned correctly.

if line Slope from sender to message is **negative**
Or
line Slope from message to receiver is **negative then**
schedule is **not** valid
end if

Algorithm 1. Slop Evaluation 1

```
while line slope from all senders to recivers is left to right
    Or
    line is Vertical  do
    schedule is valid
end while
```

Algorithm 2. Slop Evaluation 2

By finding the mentioned problems, MeSViz proved that it is a trustable solution to discover scheduling challenges. We corrected the scheduler (e.g., data structure, elements, and algorithms) and some new control conditions and constraints were added to MeSViz. The new results were re-evaluated and had shown that the scheduler and all items work correctly (e.g., Figure 1 and Figure 3).

V. CONCLUSIONS

In this article, we introduced MeSViz, a software tool that visualizes schedules and meta schedules on adaptive TT and MPSoC platforms. This tool provides a systematic way to realize a meta-visualizer regarding design, simulation, and analysis of visualizing MeS.

The primary purpose is to prepare a graphical visualizer interface tool for visualizing schedules and meta-schedules as gantt mappings at different layers and levels. It guides engineers and developers to get a fast and straightforward understandable abstract overview of behavior for different events. It is related to decision variables, constraints of the scheduling algorithm and modeling. It can be used as a scientific or educational tool to investigate how the MeS algorithms behave when an event happens and how AM, PM, and SM are changed to adapt to each situation.

MeSViz can support and display a wide range of resources and tasks for adaptive TT systems. It helps to improve the stability of platforms and to validate meta-schedules in an easy and fast way to find bugs, overlaps, and collisions in schedules and their hierarchy. It also helps to provide better maintenance even with the sudden arrival of events in the system. The schedule for each event that is generated by MeS can be shown separately and each connectivity, group of events and start time for each MeS can take a different color according to the types. MeSViz provides two distinct visualization levels of schedules for basic and meta-schedules.As a part of the future work, we will work on more scalability and parametric features to cover a broad range of problems and integrate web-based outputs and reports for online knowledge sharing and teamwork collaboration. This tool not only can be used in computer science but also for other scheduling problems that need a visualization (e.g., human resource, job scheduling, Multiple Resource–Constrained Scheduling).

ACKNOWLEDGEMENTS

The Europian project SAFEPOWER has supported this work under the Grant Agreement No. 687902.

References.

[1] R. Obermaisser, Time-triggered communication, CRC Press, 2001.
[2] M. Peter, "Visualization of scheduling in real-time embedded systems. 2013.," University of Stuttgart, Stuttgart, 2013.
[3] Hunold; Sascha; Ralf Hoffmann; Suter, Frédéric, "Jedule: A tool for visualizing schedules of parallel applications," in 39th International Conference on Parallel Processing Workshops ,IEEE, 2010.
[4] Brunst; D, Kranzlm¨uller; Muller; S., M.; Nagel, W. E., "Tools for scalable parallel program analysis: Vampir NG, MARMOT, and DeWiz," Int. J. Comput. Sci. Eng, vol. 4, no. 3, p. 149–161, 2009.
[5] A. M, "PhD THESIS: Allocation, Assignment and Scheduling for Multi-processor System on Chip," December 2006.
[6] Cortés; Alejandro, Luis ; Eles, Petru; peng, Zebo, "Quasi-Static Scheduling for Multiprocessor Real-Time Systems with Hard and Soft Tasks," in Embedded and Real-Time Computing Systems and Applications, 2005.
[7] Persya; Christy, A; Gopalakrishnan Nair, TR, "Model-Based Design of Super Schedulers Managing Catastrophic Scenario in Hard Real-Time Systems," in Information Communication and Embedded Systems (ICICES), 2013.
[8] Kopetz, H.; Bauer, G, "The time-triggered architecture," Proceedings of the IEEE, vol. 91, pp. 112-126, 2003.
[9] H. Kopetz, Real-Time Systems, Second Edition ed., Springer Science+Business Media, LLC, 2011.
[10] Murshed, Ayman ; Obermaisser, Roman; , Hamidreza,; Khalifeh, Ala; Ahmadian, Hamidreza, "Scheduling and Allocation of Time-Triggered and Event-Triggered Services for Multi-Core Processors with Networks-on-a-Chip," in In Industrial Informatics (INDIN), 2015.
[11] "ChartDirector," Advanced Software Engineering , [Online]. Available: http://www.advsofteng.com/product.html.
[12] Hoeseok; Yang; Soonhoi, Ha, "Pipelined data parallel task mapping/scheduling technique for MPSoC," in Proceedings of the conference on Design, automation and test in Europe, 2009.
[13] "XML Schema Tutorial, www.w3schools.com/Xml"
[14]M.Cumming, "The GNOME Project,"developer.gnome.org/libxml++-tutorial/stable"
[15] HAN, Gang, et al. SAFE: Security-aware FlexRay scheduling engine. In: *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*. IEEE, 2014. p. 1-4.
[16] SZILAGY, Chris; KOOPMAN, Philip. A flexible approach to embedded network multicast authentication. 2008.
[17] TANASA, Bogdan, et al. Scheduling for fault-tolerant communication on the static segment of FlexRay. In: *Real-Time Systems Symposium (RTSS), 2010 IEEE 31st*. IEEE, 2010. p. 385-394.
[18] SCHENKELAARS, Thijs; VERMEULEN, Bart; GOOSSENS, Kees. Optimal scheduling of switched FlexRay networks. In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*. IEEE, 2011. p. 1-6.
[19] MANUAL, CPLEX User's. IBM ILOG CPLEX Optimization Studio, Version 12 Release 7. 1987-2016.
[20] SCHMIDT, Ece Guran; SCHMIDT, Klaus. Message scheduling for the flexray protocol: The dynamic segment. *IEEE Transactions on Vehicular Technology*, 2009, 58.5: 2160-2169.
[21] R. Bosch GmbH, CAN Specification, Version 2, Sept. 1991.
[22] FlexRay Consortium. FlexRay Communications System Protocol Specification, Version 2.1, Revision A, December 2005.
[23] Peti, P., Obermaisser, R., Tagliabo, F., Marino, A., & Cerchio, S. (2005, May). An integrated architecture for future car generations. In *Object-Oriented Real-Time Distributed Computing, 2005. ISORC 2005. Eighth IEEE International Symposium on* (pp. 2-13). IEEE.
[24] "Flying Air Taxis from Germany Conquer the, http://press.volocopter.com/index.php/flying-air-taxis-from-germany-conquer-the-world
[25] B. Sorkhpour, R. Obermaisser and A. Murshed, "Meta-Scheduling Techniques for Energy-Efficient, Robust and Adaptive Time-Triggered Systems," in *2017 IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI)*, Tehran, 22 Dec - 22 Dec 2017.