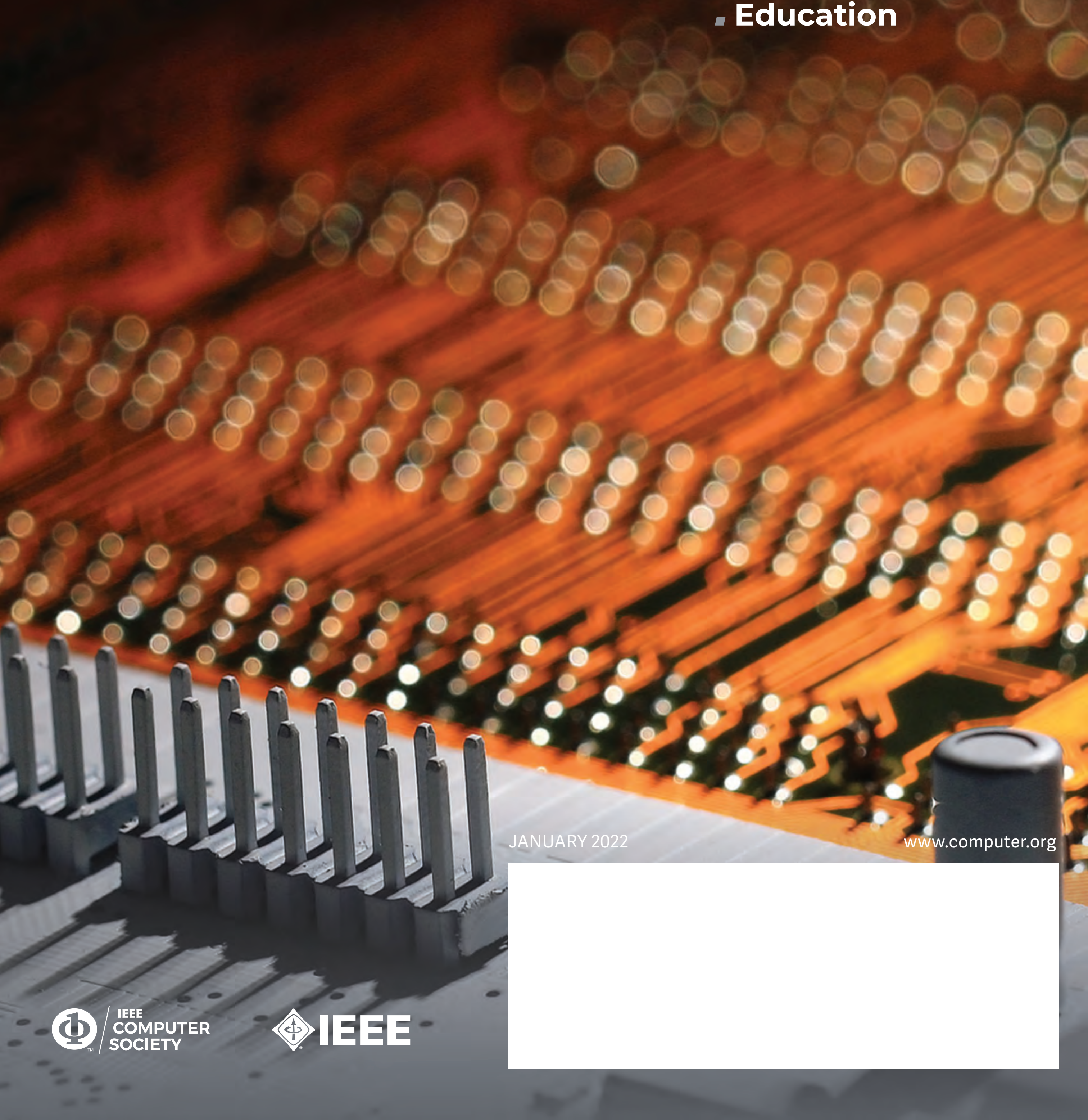
COMPUTING edge

- Embedded Systems
- Software Development
- Autonomous Vehicles
- Education



JANUARY 2022

www.computer.org

IEEE COMPUTER SOCIETY

IEEE

IEEE COMPUTER SOCIETY

Publications Seek 2023 Editors in Chief

The application deadline is 1 March 2022.

The IEEE Computer Society (IEEE CS) seeks applicants for the position of editor in chief (EIC) for several of its leading publications: *IEEE Annals of the History of Computing, IEEE Computer Graphics and Applications, IEEE MultiMedia, IEEE Pervasive Computing, IT Professional, IEEE Transactions on Affective Computing, IEEE Transactions on Dependable and Secure Computing, IEEE Transactions on Mobile Computing, IEEE Transactions on Parallel and Distributed Systems*, and *IEEE Transactions on Software Engineering*. EIC terms begin 1 January 2023.

Candidates for any IEEE CS EIC position should possess a good understanding of all aspects of the publication's field. Candidates must demonstrate the managerial skills necessary to process manuscripts through the editorial cycle in a timely fashion. An EIC must be able to attract a diverse group of talented and respected experts to their editorial board. Candidates with significant prior publications experience are preferred.

We seek applicants who are (or are willing to become) IEEE members and have clear employer support.

For more information, please go to www.computer.org/press-room/2021-news/ieee-cs-pubs-seek-2023-editors-in-chief.

Submit an application today!

IEEE COMPUTER SOCIETY

IEEE

STAFF

Editor
Cathy Martin

Publications Operations Project Specialist
Christine Anthony

Production & Design Artist
Carmen Flores-Garvey

Publications Portfolio Managers
Carrie Clark, Kimberly Sperka

Publisher
Robin Baldwin

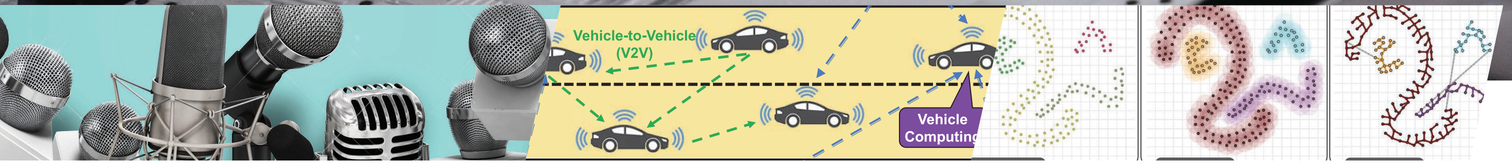
Senior Advertising Coordinator
Debbie Sims

Circulation: *ComputingEdge* (ISSN 2469-7087) is published monthly by the IEEE Computer Society. IEEE Headquarters, Three Park Avenue, 17th Floor, New York, NY 10016-5997; IEEE Computer Society Publications Office, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720; voice +1 714 821 8380; fax +1 714 821 4010; IEEE Computer Society Headquarters, 2001 L Street NW, Suite 700, Washington, DC 20036.

Postmaster: Send address changes to *ComputingEdge*-IEEE Membership Processing Dept., 445 Hoes Lane, Piscataway, NJ 08855. Periodicals Postage Paid at New York, New York, and at additional mailing offices. Printed in USA.

JANUARY 2022 · VOLUME 8 · NUMBER 1

COMPUTING edge



14 Formal Verification of Cyberphysical Systems

34 The Emergence of Vehicle Computing

42 Teaching Clustering Algorithms With EduClust: Experience Report and Future Directions

Embedded Systems

8 Determining Reliable and Precise Execution Time Bounds of Real-Time Software
REINHARD WILHELM

14 Formal Verification of Cyberphysical Systems
JAMES BRET MICHAEL, DORON DRUSINSKY, AND DUMINDA WIJESEKERA

Software Development

26 Boris Cherny on TypeScript
NATE BLACK

30 Toward Unseating the Unsafe C Programming Language
PAUL C. VAN OORSCHOT

Autonomous Vehicles

34 The Emergence of Vehicle Computing
SIDI LU AND WEISONG SHI

40 Secure V2V and V2I Technologies for the Next-Generation Intelligent Transportation Systems
SUDIP MITTAL

Education

42 Teaching Clustering Algorithms With EduClust: Experience Report and Future Directions
JOHANNES FUCHS, PETRA ISENBERG, ANASTASIA BEZERIANOS, MATTHIAS MILLER, AND DANIEL A. KEIM

Departments

4 Magazine Roundup
7 Editor's Note: Embedded Systems Software
50 Conference Calendar

Magazine Roundup

The IEEE Computer Society's lineup of 12 peer-reviewed technical magazines covers cutting-edge topics ranging from software design and computer graphics to Internet computing and security, from scientific applications and machine intelligence to visualization and microchip design. Here are highlights from recent issues.

Computer

Explainable Recommendations and Calibrated Trust: Two Systematic User Errors

The increased adoption of collaborative human-artificial intelligence decision-making tools triggered a need to explain recommendations for safe and effective collaboration. The authors of this article from the October 2021 issue of *Computer* explore how users interact with explanations and why trust-calibration errors occur, taking clinical decision-support systems as a case study.

Computing in Science & Engineering

Performance Portability in the Exascale Computing Project: Exploration Through a Panel Series

Performance portability is a critical issue for the Exascale Computing Project (ECP) because of nontrivial architectural differences between machines available today and those expected at exascale. Many ECP project teams are working toward performance portability and would expect to benefit from sharing lessons learned, identifying gaps, and discovering opportunities for partnerships. To facilitate this communication, the IDEAS-ECP project partnered with the three focus areas of ECP (application development, software technology, and hardware and integration) and Department of Energy computing facilities to lead a series of panel discussions on performance portability. The panels were organized around broadly common themes of algorithmic and data locality challenges. In this article from the September/October 2021 issue of *Computing in Science & Engineering*, the authors describe the panel series, its objectives, and perspectives from the various areas of the project.

IEEE Annals of the History of Computing

Theoretical Computer Science in Italy: The Early Years

In this article from the July–September 2021 issue of *IEEE Annals of the History of Computing*, the authors provide an overview of the early developments of theoretical computer science research in Italy in the 1960s and early 70s. The community of researchers working in this domain were organizing to gain an identity among the more traditional disciplines and to obtain a recognition for the "new science" that was taking its first steps. This led in Italy to the creation of the Group of Researchers in Theoretical Informatics, in parallel to the institution of the European Association for Theoretical Computer Science at the European level.

IEEE Computer Graphics and Applications

A Visual Analytics Approach for Structural Differences Among Graphs via Deep Learning

Representing and analyzing structural differences among graphs present insight into difference-related patterns such as dynamic evolutions of graphs. Conventional solutions leverage representation learning techniques to encode structural information but lack an intuitive way of studying structural semantics of graphs. In this article from the September/October 2021 issue of *IEEE Computer Graphics and Applications*, the authors propose a representation-and-analysis scheme for structural differences

among graphs. They propose a deep learning-based embedding technique to encode multiple graphs while preserving semantics of structural differences.

IEEE Intelligent Systems

Fuzzy Graph and Collective Multiagent Reinforcement Learning for Traffic Signals Control

Multiagent systems provide proper modeling in real-world applications such as intelligent transportation systems. The interaction between the agents can be represented by the graph theory. In this article from the July/August 2021 issue of *IEEE Intelligent Systems*, a fuzzy graph is used for urban traffic network modeling. A network composed of several intersections is considered a multiagent system composed of multiple interacting agents. The interaction between the agents can be represented by a fuzzy graph in which each vertex shows an agent in the network. The network is divided into correlated agent's sets. In each set, collective learning composed of Q-learning and function approximation method is used to learn the optimal control policy. The total average energy of the sets of correlated agents as fuzzy subgraphs is computed, and the relationship between these values and the effectiveness of the collective learning is studied.

IEEE Internet Computing

Autonomics at the Edge: Resource Orchestration for Edge Native Applications

With the increasing availability of edge computing resources, there is a need to develop edge orchestration and resource management techniques to support application resilience and performance. Similar to the use of containers and microservices for cloud environments, it is important to understand the key attributes that characterize edge native applications. As edge devices increase in their autonomy and intelligence, orchestration techniques are needed to respond to changes in device properties, availability, security credentials, migration, and network connectivity protocols. Implementing autonomics techniques for edge computing can increase the resilience of the interaction between devices and applications, reducing execution time and cost. Read more in this article from the July/August 2021 issue of *IEEE Internet Computing*.

IEEE micro

Quantum Codesign

Codesign has been an integral part of computer architecture since the very first systems were brought online. From the early days of the field until now, end-user applications inevitably shape the design and capabilities of subsequent generations of hardware. Likewise, the characteristics and capabilities of new computational hardware and systems often impact the algorithms and software that run on them. Quantum computing (QC) is similarly reliant on codesign approaches, particularly now in its resource constrained early days. This article from the September/October 2021 issue of *IEEE Micro* discusses what codesign means in a QC setting, gives examples of its value to QC, and proposes key attributes of QC codesign approaches going forward.

IEEE MultiMedia

Class-Balanced Text to Image Synthesis With Attentive Generative Adversarial Network

Although the text-to-image synthesis task has shown significant progress, generating high-quality images remains a challenge. In this article from the July–September 2021 issue of *IEEE MultiMedia*, the authors first propose an attention-driven, cycle-refinement generative adversarial network, AGAN-v1, to bridge the domain gap between visual contents and semantic concepts by constructing spatial configurations of objects. Second, an advanced class-balanced generative adversarial network, AGAN-v2, is proposed to address the problem of long-tailed data distribution. Importantly, it is the first method to solve this problem in the text-to-image synthesis task.

IEEE pervasive COMPUTING
MOBILE SYSTEMS | UBIQUITOUS COMPUTING | INTERNET OF THINGS

Hybrid Body Craft: Toward Culturally and Socially Inclusive Design for On-Skin Interfaces

Sensor device miniaturization and breakthroughs in novel materials have enabled technology to progress directly onto the skin surface. However, unlike all other media, the human body is a complex and meaning-laden surface that encompasses a wearer's individual, social, and political identities. Yet, research in on-skin interfaces has focused on engineering aspects, with a scant focus on the cultural and social dimensions of device design. Hybrid Body Craft presents a design approach for bridging the cultural aspects of body crafts with emerging on-skin interfaces. The authors of this article from the July–September 2021 issue of *IEEE Pervasive Computing* present a series of more socially and culturally inclusive on-skin interface designs that incorporate various emerging materials and technologies into body craft customs.

IEEE SECURITY & PRIVACY

Demystifying Android's Scoped Storage Defense

Android recently introduced the scoped storage defense to better protect application use of shared external storage. This article from the September/October 2021 issue of *IEEE Security & Privacy* examines the evolution of Android external storage defenses leading to scoped storage and assesses the impact of the scoped storage defense for limiting opportunities for exploitation.

IEEE Software

What Do We Know About Time Pressure in Software Development?

Time pressure means that time experienced by an individual is scarce in relation to the task demands at hand. In this article from the September/October 2021 issue of *IEEE Software*, the authors summarize findings and provide practitioner takeaways based on a systematic review of existing literature.

IT Professional

Healthcare Insurance Frauds: Taxonomy and Blockchain-Based Detection Framework (Block-HI)

Medical health insurance fraud has been a major concern for the healthcare industry and governmental institutions. In the United States, health insurance companies record a loss of tens of billions yearly due to healthcare fraud. Some types of fraud are at the risk of the patient's health. This is because the system that performs the manual processing of medical insurance claims frequently misses the endorsement of some stakeholders (such as the patient, pharmaceutical companies, wholesale dealers, and medical equipment suppliers) in a claim's validation process. Blockchain is a peer-to-peer distributed system that can enable the validation of healthcare claims in a secure, immutable, and transparent manner. The authors of this article from the July/August 2021 issue of *IT Professional* present a taxonomy of healthcare insurance claims frauds and propose a blockchain-based healthcare insurance claims fraud detection framework.

Embedded Systems Software

Software in embedded systems is often time- and safety-critical, so dependability is essential. For cyberphysical systems such as autonomous vehicles, it's important to be able to verify the software's security and reliability. Two articles in this issue of *ComputingEdge* explore ways to improve verification of software in embedded systems.

IT Professional's "Determining Reliable and Precise Execution Time Bounds of Real-Time Software" stresses the importance of robust timing verification for embedded control system software in transportation, medicine, and manufacturing. *Computer*'s "Formal Verification of Cyberphysical Systems" brings together seven experts to discuss a promising verification approach for embedded systems software.

The programming language used in software development can either help or hurt the quality of the software. "Boris Cherny on TypeScript," from *IEEE Software*, presents an interview about a scalable programming language aimed at catching mistakes. In "Toward Unseating the Unsafe C Programming Language," from *IEEE Security & Privacy*, the author makes the case for using modern languages with built-in safety properties.

Tomorrow's self-driving cars will be more connected than ever. The authors of *IEEE Internet Computing*'s "The Emergence of Vehicle Computing" describe their vision of autonomous vehicles that communicate with one another and with the environment. *Computer*'s "Secure V2V and V2I Technologies for the Next-Generation Intelligent Transportation Systems" highlights a model for securing smart car communication.

This *ComputingEdge* issue closes with new ideas in education. The authors of "Teaching Clustering Algorithms With EduClust: Experience Report and Future Directions," from *IEEE Computer Graphics and Applications*, recount using an online visualization application to teach clustering algorithms.

This article originally appeared in IT Professional vol. 22, no. 3, 2020

Determining Reliable and Precise Execution Time Bounds of Real-Time Software

Reinhard Wilhelm, *Saarland University and AbsInt Angewandte Informatik*

TIME-CRITICAL EMBEDDED SYSTEMS

Embedded-control systems in transportation, medical instruments, and manufacturing are often time critical, that is, they have to finish execution within their period or they have to respond to sensor input within a given deadline. Both period and deadline are dictated by the physics of the system. Examples for sensor-actuator control are the airbag and the automatic brake control in cars. Clearly, if the airbag controller fails to inflate the airbag in time, the driver will hit the steering wheel. Flight control and guidance, collision-avoidance, or ground proximity warning systems in airplanes are examples from the aviation domain. Deadlines in airplanes are often in the order of milliseconds, while crankshaft-synchronous tasks in cars often have periods on the order of microseconds.

A rigid verification of time-critical software should include a static timing verification. Measurement-based methods cannot guarantee to hit the worst case and to determine its execution time (WCET) because the number of cases to consider is too large to allow exhaustive methods. Certification rules and practice of the European Union Aviation Safety Agency (EASA) requires the use of static timing analysis in the certification of time-critical plane components.

PROBLEM

The general setting for WCET analysis is that a set of hard real-time tasks is to be executed on a given hardware platform. Hard real-time tasks have associated deadlines within which they have to finish their execution. The deadlines may be given by periods. *Timing Verification* has to verify that these timing constraints are satisfied.

Traditionally, Timing Verification is split into a *WCET analysis*, which determines upper bounds on the execution times of all tasks, and a *schedulability analysis*, which takes these upper bounds and attempts to verify that the given set of tasks when executed on the given platform will all respect their deadlines.

The introduction of performance-enhancing architectural components and features, such as caches, pipelines, and speculation, made the execution time of instructions dependent on the architectural state in which they are executed, essentially the occupancy of resources. The execution times of instructions now vary largely. For example, an instruction accessing memory may last a few cycles but also up to several hundred cycles. The variability of execution times grew with several architectural parameters, e.g., the cache-miss penalty and the costs for pipeline stalls and for control-flow mis-predictions. For these architectures the determination of WCETs became difficult. We started research into the WCET-analysis problem in the middle of the 90s.

The introduction of multicore execution platforms into the embedded real-time domain made the problem still more difficult. These platforms typically have shared resources, and the interference on these shared resources complicates the determination of upper execution-time bounds for tasks executed on such a platform.

A few words about terminology: From the beginning, we aimed at *sound* WCET-analysis methods. The results of a sound WCET analysis are *conservative*, i.e., they will never be exceeded by an execution. We

Digital Object Identifier 10.1109/MITP.2020.2972138
Date of current version 21 May 2020.

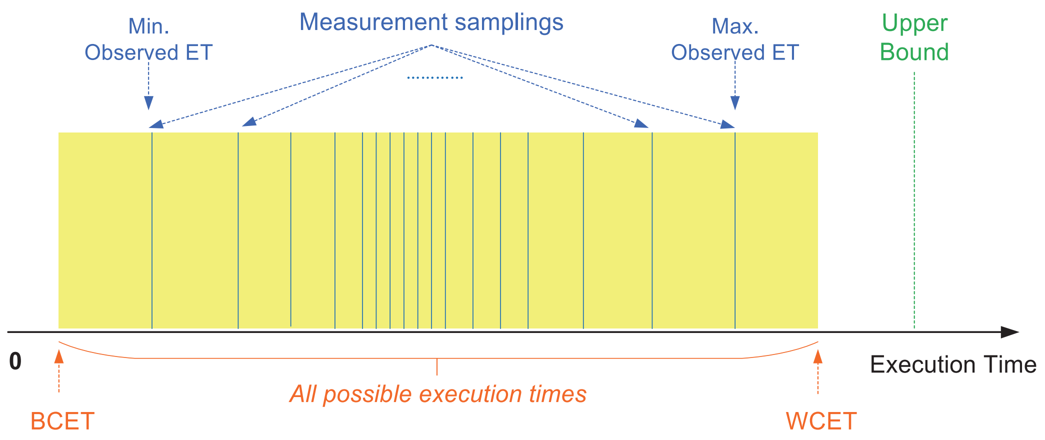


FIGURE 1. Upper bound of all execution times is computed instead of the real WCET.

consider being conservative as a Boolean property. Often, *conservative* is used as a metric, being more conservative meaning being less accurate. For an unsound, e.g., measurement-based method, it does not make sense to speak about being more or less conservative. Such a method may under-estimate or over-estimate the real WCET. So, being "more conservative" may mean moving toward the real WCET from below, or it may mean moving further away from the real WCET by increasing over-estimation. Besides soundness, the second, quite important property is *accuracy* of the results of a WCET analysis. For a sound WCET analysis method, accuracy corresponds to the degree of over-estimation.

The execution of a program can be seen as one particular walk through its call- and control-flow graphs: branches in the graph taken, depending on input values. There may be many such walks. However, there are even more different paths through the architecture, depending on the execution state of the architecture, essentially the occupancy of resources. A memory access may hit or miss the cache, resulting in at least two different paths; in the cache-miss case, there may be again at least two different continuations, depending on whether the bus is occupied or not. WCET-analysis can be seen as the search for a longest path in the state space spanned by the potential paths through the program and by the potential paths through the architectural platform.

WCET analysis is based on the assumption that the analyzed programs terminate, that is, all recursions and iterations are bounded. We are not confronted with the undecidability of the halting problem. Our tool might discover that it cannot determine bounds on recursion or iteration in a program and will ask the user for annotations. All WCET bounds are then valid with respect to the given annotations. This state space is thus finite but too large to be exhaustively explored. The implementation of the technology is a permanent fight with the complexity of the task. Safe over-approximation is used in several places (Figure 1). In particular, an abstraction of the execution platform is employed by the WCET analysis.

Accuracy: For a sound tool, i.e., one that never under-estimates the WCET, accuracy is defined as the degree of over-estimation. Over-estimation is the difference between the upper bound computed by the tool and the worst ever observed execution time. 15–25% over-estimation was reported for Airbus code in.[8]

PROBLEMS TO BE SOLVED

At the core of the WCET-analysis problem was the notion of a *Timing Accident*: We understand it to be any architectural effect that lets an instruction execute longer than its fastest execution time. Examples for such timing accidents are cache misses, pipeline stalls, bus-access conflicts, and branch mispredictions. Each such timing accident has to be paid for, in terms of execution-time cycles, by an associated

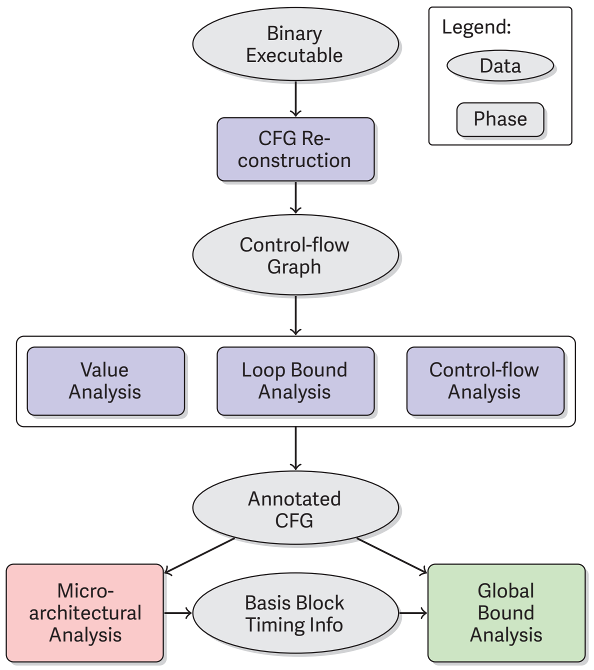


FIGURE 2. Tool architecture.

Timing Penalty. The size of a timing penalty can be constant but may also depend on the execution state. We consider the property that a particular instruction will not cause a particular timing accident as a *safety property*. The occurrence of a timing accident thus violates a corresponding safety property.

These fundamental notions led us to our approach.

- Use an appropriate method for the verification of safety properties to prove that for individual instructions in the program, some of the potential timing accidents will never happen. Reduce the worst-case execution-time bound for an instruction, which a sound WCET analysis would have to assume, by the penalties for the excluded timing accidents.
- Abstract interpretation[1] is a powerful method to prove safety properties. Use it to compute certain invariants at each program point, namely an upper approximation of the set of execution states that are possible when execution reaches this program point.
- Derive safety properties, that certain timing accidents will never happen, from these invariants. This method for the microarchitectural analysis was the central innovation that made our WCET analysis work and scale.

Given this understanding, our task was to design abstract domains for the analysis of the behavior of several relevant architectural components and combine them in a sound way.

This meant we had to define

- *domains of abstract states* with a *partial order* representing which domain elements contained better information than other elements;
- corresponding to this partial order, a *join function*, used to combine incoming abstract domain elements at control-flow merge points;
- *abstract effects* for each instruction, describing the update of abstract states corresponding to this instruction.

I cannot stress enough that this systematic design of the instances of our timing-analysis technology for different architectures was essential for our success. It opened a way to use a generic approach, which was essential for the instantiation of the technology for new architectures. Our competition in this research field typically presented a page of pseudo-C code and a claim that this would be a solution to, say, cache analysis. It was next to impossible to give correctness arguments for this claim.

I will explain our timing-analysis approach along Figure 2. Timing analysis has to be performed on the binary-executable level because the actually executed program is what matters for execution time. Compiler optimizations, memory, and register allocation may have a huge impact on timing. Fully linked binary executables have to be decoded, and the control flow of the program has to be reconstructed. These are nontrivial, compiler-dependent tasks. Jump tables and the nonexistence of function-return instructions are the most complex issues.

A number of static analyses are performed on the resulting control-flow graph. The first one, called *Value Analysis*, propagates information about statically known values in program variable and registers. It essentially is an *Interval Analysis* computing intervals for these variables and registers at each program point. These intervals enclose all potential values the variables or registers may have when execution reaches this program point. This information is absolutely needed for data-cache analysis. Without it, one would rarely know where a data access would go in memory. Value Analysis often also determines loop bounds. Such bounds are needed to arrive at bounded execution times. The user is asked to supply loop bounds in case they cannot be statically determined. *Control-Flow Analysis* also exploits results of Value Analysis to get rid of infeasible control-flow paths. Such infeasible paths may dilute the accuracy by contributing execution times that are, in fact, not possible. The control-flow graph is annotated with the results of these three static analyses.

At the heart of our timing-analysis technique is the *Microarchitectural Analysis*. It performs what I have sketched above; namely, it computes invariants at all program points that describe all states of platform components that are possible when execution reaches this program point, and it finds out which timing accidents cannot happen. The results are upper bounds for the execution time of all basic blocks. Details about the analyses of caches, pipelines, and system controllers can be found in[3–5]. The overall approach is described in.[6]

The control flow is translated into an integer linear program (ILP) roughly according to what Li and Malik have proposed.[7] Additional information about the control flow can often be encoded into the ILP to increase accuracy. This information may originate from user annotations. The solution of this ILP gives a longest path through program and architecture and an associated execution-time bound.

BREAKTHROUGH

We were lucky to receive consecutive research grants from DFG, our National Science Foundation, and the European Union. The Daedalus project, funded by the EU, brought us together with an extremely valuable partner, Airbus, who was searching for a solution for their WCET-analysis problem. They knew that their previously used measurement-based method, also used in certification, did not work any longer for the execution platform selected for the Airbus A380, namely the Motorola MPC755. The Airbus people kept us focused on the real problems, including the analysis of peripheries and system controllers: architectural components that WCET researchers had never considered.

They provided us with benchmark software, a set of 12 denatured tasks, each consisting of several million instructions, as they were flying them in the A340. The tool we developed until 2001 was able to analyze the benchmark provided by Airbus in decent time and with quite precise results. The upper bounds it computed made the Airbus people quite happy because they were roughly in the middle between the worst observed execution times and the upper bound determined by Airbus with a measurement-based method using safety margins. More precisely, our analysis results were overestimating the worst observed execution times by roughly 15%. This breakthrough was reported in[2]. This article received the EMSOFT 2019 Test-of-Time Award.

As a result of our successful development, Airbus offered our tools to the certification authorities for the certification of several Airbus plane generations, starting with the Airbus A380. The European Union Aviation Safety Agency (EASA) has accepted the AbsInt WCET analysis tool as validated tool for several time-critical subsystems of these plane types. We were less successful with Airbus' competitor, who partly certifies their planes themselves, as it recently turned out, and with the certification authority in charge, who does not seem to require the use of a sound verification technology for real-time requirements.

ABSINT AND THE INDUSTRIALIZATION OF WCET TECHNOLOGY

We had essentially solved the WCET-analysis problem for single-core architectures in a sequence of Ph.D. theses in my group. The only import was the Implicit Path Enumeration Technique (IPET) of Li and Malik.[7] However, there is more to it when a practically usable tool is required. WCET analysis consists of many phases. A practically usable WCET-analysis method requires strong solutions to all the subproblems and their adequate interaction. Otherwise, either the effort is too high, or the accuracy is too low.

The people at AbsInt did an excellent engineering job to come up with WCET-Analysis tools and later, also, other tools that were usable on an industrial scale.

Costs: There are several different types of costs to be considered:

1. The development of the WCET-analysis technology cost roughly 20 person years.
2. The instantiation for a new platform may cost several person years depending on the complexity of the platform.
3. Most customers buy licenses because they keep their safety-critical code in-house.
4. WCET-analysis service can be bought by paying for working days and tool rent.

Which were the main challenges and which were the important engineering principles in the design of aiT: the AbsInt WCET-analysis tool? The main challenge was the complexity of several subtasks. One of the important engineering principle was *modularization* in several versions: first, an adequate separation of analysis phases and, within individual phases, the separation of generic parts and architecture-specific parts to ease the instantiation for new architectures. Value Analysis was modularly composed of several analysis domains such that new domains could be added if necessary. Complexity reduction required the selection of the most efficient data structures for the analysis domains, including decisions that information to determine on demand instead of storing it. Soundness of the overall analysis critically depends on the abstract machine model being conservative. Validation of this property is a time-consuming process required by the certification authorities. The core of this process is *trace validation*. It uses the abstract machine model as a predictor of traces and then compares observed traces with predicted traces. It is described in detail in[9].

We had founded AbsInt to industrialize our WCET technology. We had solved the problem; we had instantiations for some processor architectures, basically for those that Airbus and their suppliers needed. However, we had to learn that hardly any two potential customers employed the same architecture configuration. The decision for a new platform was taken without considering whether a WCET-analysis existed for this platform. Instantiating our technology for a new, complex platform takes a lot of effort, and platforms were not getting simpler! In consequence, such an instantiation is very expensive, which does not raise the motivation of potential customers to buy our WCET tools or order the development of a new instance for their platform. In fact, the insight that verification of real-time behavior was necessary was not too widespread. There existed also some competitors, who marketed their measurement-based, unsound timing analysis and often forgot to mention the unsoundness of their tool. They could offer their tools at much cheaper prices. So, industrializing and marketing a sound WCET technology, that inherently needed to be expensive, was not a promising way to get rich. However, our development of a sound method that actually solved a real problem of the safety-critical industry was considered a major success story for the often-disputed formal-methods domain. AbsInt became the favorite partner for the industrialization of academic prototypes. Patrick Cousot and his team offered their prototype of Astrée, a static analysis for run-time errors, which in cooperation with some of the developers has been largely extended by AbsInt. Then, Xavier Leroy offered the result of his much acclaimed research project CompCert: the first verified optimizing C compiler. Both Astrée and CompCert are now AbsInt products.

The AbsInt WCET-analysis tool aiT is offered for quite a range of architectures, e.g., a large range of PowerPCs, several versions of the TriCore and the ARM architectures, the LEON2 and LEON3, in combination with quite a range of compilers. For architectures with bad timing predictability, the TimeWeaver tool computes execution-time estimates. It is based on measuring execution traces.

The users of model-based design tools appreciate tight integrations of aiT with code generators. This integration provides for the exploitation of model information and the back annotation with performance figures to the model. Such integrations have been done for the SCADE Suite, targetlink, ASCET, and others.

REFERENCES

1. P. Cousot and R. Cousot, “Abstract interpretation: A unified lattice model for static analysis of programs by

construction or approximation of fixpoints," in *Proc. ACM Princ. Program. Lang.*, 1977, pp. 238–252.
2. C. Ferdinand, et al., "Reliable and precise WCET determination for a real-life processor," in *Proc. Int. Workshop Embedded Softw.*, 2001, pp. 469–485, LNCS 2211, Springer.
3. C. Ferdinand and R. Wilhelm, "Efficient and precise cache behavior prediction for real-time systems," *Real-Time Syst.*, vol. 17, no. 2/3, pp. 131–181, 1999.
4. M. Langenbach, S. Thesing, and R. Heckmann, "Pipeline modeling for timing analysis," in *Proc. Int. Static Anal. Symp.*, 2002, pp. 294–309, LNCS 2477, Springer.
5. S. Thesing, "Modeling a system controller for timing analysis," in *Proc. 6th ACM Embedded Softw.*, 2006, pp. 292–300.
6. R. Wilhelm, et al., "Static timing analysis for hard real-time systems," in *Proc. Int. Workshop Verification, Model Checking Abstract Interpretation*, 2010, pp. 3–22.
7. Y.-T. S. Li and S. Malik, "Performance analysis of embedded software using implicit path enumeration," in *Proc. 32nd ACM/IEEE Des. Autom. Conf.*, 1995, pp. 456–461.
8. S. Thesing, et al., "An abstract interpretation-based timing validation of hard real-time avionics software," in *Proc. Int. Conf. Dependable Syst. Netw.* 2003, pp. 625–632.
9. R. Wilhelm, M. Pister, G. Gebhard, and D. Kästner, "Testing implementation soundness of a WCET analysis tool," in *Festschrift in Honor of Peter Marwedel*. Berlin, Germany: Springer LNCS, to be published.

REINHARD WILHELM is a professor emeritus at the Informatics Department of Saarland University in Saarbrücken, Germany. He has been the Scientific Director of the Leibniz Center for Informatics in Schloss Dagstuhl from 1990 to 2014. He is a cofounder of the spin-off company AbsInt Angewandte Informatik, located in The Science Park in Saarbrücken, Germany. He is a Fellow of the ACM. Contact him at wilhelm@cs.uni-saarland.de.

COLUMN: VIRTUAL ROUNDTABLE

This article originally appeared in **Computer** vol. 54, no. 9, 2021

Formal Verification of Cyberphysical Systems

James Bret Michael and Doron Drusinsky, *Naval Postgraduate School*

Duminda Wijesekera, *George Mason University*

Computer *hosts a virtual roundtable with seven experts to discuss the formal specification and verification of cyberphysical systems.*

In *Computer*, virtual roundtables (VRTs) are virtual panels. We ask a series of questions to a group of experts via email to ascertain the panelists' thoughts about a topic du jour. One difference between VRTs and face-to-face panels is that no expert knows who the others are. That is different from an in-person arrangement, where answers from one participant can affect the responses of others. In this VRT, our topic of discussion is the formal verification (FV) of cyberphysical systems (CPSs). FV is the act of proving the correctness of algorithms with respect to certain formal specifications, using formal methods. *Correctness* may mean logical definitions of safety, liveliness, and other objectives such as confidentially, integrity, availability, and some version of privacy.

FV has its roots in formal reasoning, dating back at least to Gottfried Wilhelm Leibniz's work on algorithms, computing machines, and mathematical logic in the 17th century.[1] FV as we know it has its roots in the 1960s and 1970s with the contributions of E.W. Dijkstra, who famously coined the phrase, "Program testing can be used to show the presence of bugs, but never to show their absence."[2] Some believed FV to be a silver bullet for attaining dependable software and hardware. The excitement over formal methods is evidenced by the relatively large body of published research on the topic; a Google scholar search for the FV term yields approximately 180,000 results.

Digital Object Identifier 10.1109/MC.2021.3055883
Date of current version: 27 August 2021

Nevertheless, except for a few well-funded research projects, industry was rather slow to adopt FV. An exception to this statement is the semiconductor design community, also known as the electronic design automation (EDA) community. This community realized that the cost and delays incurred by labor-intensive manual testing justified a different verification approach, one that applied FV. Moreover, since manual testing cannot guarantee the absence of bugs, there is an inevitable cost for containing the impact of flaws that are undetected. A classic example is the Intel Pentium FDIV bug, which was difficult for testing to uncover: containment required replacing all flawed Pentium processors on request, with Intel taking a US$475 million charge against earnings.[3,4] Fast forward to 2021, and the EDA community has embraced FV as part of mainstream development and verification processes, languages, and environments. For example, formal specification is an integral part of the SystemVerilog IEEE standard.[5]

Two of the moderators for this roundtable (Michael and Drusinsky) were authors of an article that recommended applying lightweight formal methods to the interfaces between the cyber and physical parts of a CPS.[6] This recommendation, along with our discussion of open questions in formal methods, drew a lot of interest; for instance, see the exchange between the authors and Michael Jackson.[7] The feedback we received from our counterparts in the formal methods community, in combination with the third moderator's (Wijesekera's) experiences in applying formal methods to software-intensive systems, inspired the three of us to organize a roundtable in which we enlisted seven

experts to identify the reasons for the slow adoption of FV by the software industry, in general, and the verification of CPSs, in particular.

The panelists contend there are several factors that have slowed the adoption of formal methods, such as the sheer size and complexity of software systems, the diversity of software products, the perception that FV is a low-return-on-investment academic exercise, and the fact that FV tools are not part of mainstream software development and testing environments nor are the tools directly associated with mainstream programming languages. The FV of CPSs is believed to be particularly challenging because it is a hybrid on many fronts, including hardware and software, classical control and logical reasoning, and artificial intelligence (AI)/machine learning (ML) algorithms and logical reasoning.

In this VRT, the panelists responded to six questions. Their written responses may have undergone minor edits. However, as organizers, we attempted to keep their words as verbatim as possible. The seven panelists are Knut Åkesson (Chalmers University), Dimitra Giannakopoulou (NASA), Klaus Havelund (Jet Propulsion Laboratory), Sayan Mitra (University of Illinois at Urbana–Champaign), Corina Pasareanu (KBR), Sanjit A. Seshia (University of California, Berkeley), and Oleg Sokolsky (University of Pennsylvania). See "Roundtable Panelists" for the participants' biographical sketches. Note that the opinions of the experts are their own, with no input from the editors. We hope readers who are concerned with the dependability and trustworthiness of CPSs will find the questions and responses enlightening.

Computer: Unlike with EDA, in which FV is well integrated into the development of chips and printed circuit boards and where engineers with expertise in the method are in high demand, FV has had much less acceptance as a mainstream ingredient of software development and quality assurance. What do you think are the reasons for that, and do you think the situation will change with CPS projects, such as those involving autonomous vehicles?

Knut Åkesson: A major challenge is that the closed-loop model is described using a combination of tools, different modeling languages, and programming languages. Significant efforts have been made to unify how to describe physical systems coherently. For example, the Modelica language (https://modelica.org/modelicalanguage.html) is an essential step in this direction. However, CPSs might also contain ML algorithms for perception and might run optimization for decision making. These are all rapidly evolving and have their dedicated languages and tools. Thus, CPSs inherently combine code written in various programming languages, ML frameworks, optimization modules, and control logic generated from high-level modeling languages. FV has its place in safety-critical components but should be complemented by rigorous automated test methods for situations where it is not feasible or practical to use.

A MAJOR CHALLENGE IS THAT THE CLOSED-LOOP MODEL IS DESCRIBED USING A COMBINATION OF TOOLS, DIFFERENT MODELING LANGUAGES, AND PROGRAMMING LANGUAGES.

Dimitra Giannakopoulou: Software development is more diverse and evolves faster than EDA in terms of programming languages, paradigms, and patterns; data structures, algorithmic approaches, and types of applications; libraries and runtime environments; and heterogeneity and distribution across different computers. After deployment, software applications get updated to address vulnerabilities and to include new features, and they may even be adaptive by design. Correctness criteria and specifications vary widely by application domain, and quality assurance depends on the criticality of software. For example, is it a game on someone's phone, or is it software that controls a passenger aircraft?

For FV to become a mainstream ingredient of software development, it must achieve some usability goals. First, it must be relatively easy to formulate specifications for the target system. Second, FV must be able to directly handle the languages in which the software is written or the modeling languages from which the software is synthesized. Finally, FV should be able to scale. The diversity and complexity of software applications means that to be successful, FV approaches must be targeted and customized to address specific problems within safety-critical application domains.

The expected exponential rate of introduction of autonomous vehicles (ground and air) puts enormous pressure on ensuring their safe operation. There is incentive for commercial and federal stakeholders to collaborate on developing certification and assurance standards for these applications. As a consequence, I expect advances in the near future. On the one hand, FV approaches will be developed that efficiently address specific problems of such CPSs. On the other hand, there will be increased incentives in CPS projects to use programming paradigms and environments designed with FV in mind.

Klaus Havelund: Electronics designs have the characteristic that once they leave the factory, they usually cannot be changed. A substantial error can cause a unit to be recalled, with large amounts of money at stake. The motivation is therefore high to "get it right" before shipment. Software, on the other hand, can often be fixed with an update at a customer's location, making errors less catastrophic. Even in space missions, errors can be corrected by uplinking bug fixes from a distance of millions of miles. This relaxed view of software errors might, however, be changing as software, to an increasing extent, autonomously controls equipment such as cars, which can cause loss of life in case of failure.

Another, perhaps more important, reason for the lesser acceptance of FV in the software community is that the verification problem appears less tractable for software systems, due to higher complexity and the possibility of more execution paths. Theorem provers require a considerable amount of manual effort to apply, even for smaller models, let alone real-world software systems, and model checkers are challenged by the large state spaces of realistically sized software applications. This means that the application of FV techniques requires either a big verification effort or a big modeling effort, where a simplistic model is created of the software and then verified. A software engineer, not supported by management to carry out such proofs/modeling, will see very little incentive to do so.

Sayan Mitra: FV is being used in mainstream software already, propelled first by major outages and breaches at big tech firms, then by successful applications of verification technology in bug finding, and more recently in the application of verification for generating proofs as "more extensive tests." Static analysis tools are part of the core developer workflow at Google and deployed on the 2-billion-line code base.[8] Amazon Web Services (AWS) developers are writing formal specifications and proofs for hypervisors, boot loaders, and Internet of Things operating systems.[9] The Infer static analysis engine is integrated with the code base at Facebook and does continuous reasoning on iOS, Android, and Instagram and WhatsApp applications. Hundreds of bugs are reported and fixed every month.[10] Bugs in CPSs and autonomous systems can compromise safety. This raises the stakes as well as the incentives for the adoption of FV. But the adoption of CPS verification also presents barriers that were not present in the software ecosystem.

Corina Pasareanu: The reason is that FV for software is much harder (for example, programs are much larger, potentially unbounded, use many external libraries, and contain programming language constructs that are hard to analyze). Yes, CPS projects are often safety critical and justify the high cost of FV. Furthermore, the software involved in CPS projects is simpler than general-purpose software and therefore more amenable to verification.

ROUNDTABLE PANELISTS

Knut Åkesson is a professor of automation in the Department of Electrical Engineering, Chalmers University of Technology, Göteborg, Sweden. His research interests include rigorous methods for verification and control with applications in safety-critical autonomous systems, the optimization and configuration of products and production systems with high variability, and applications of computer vision and deep machine learning. Åkesson received a Ph.D. from Chalmers University of Technology in 2002. Contact him at knut.akesson@chalmers.se.

Dimitra Giannakopoulou is a research computer scientist at the NASA Ames Research Center, Mountain View, California, USA, and a member of the Robust Software Engineering Group. Her research interests include applying modular and compositional formal verification techniques to autonomous systems and architectures. Giannakopoulou received a Ph.D. in computer science from Imperial College, University of London, in 1999. Contact her at dimitra.giannakopoulou@nasa.gov.

Klaus Havelund is a senior research scientist at Jet Propulsion Laboratory, Pasadena, California, USA, specifically in the Laboratory for Reliable Software. His research interests include the development of runtime monitoring techniques, including the design of powerful monitoring logics. He is the chair of the Formal Methods Europe industry committee, a member of International Federation for Information Processing 1.9/2.15 working group, and a member of the *Transactions on Foundations for Mastering Change* editorial board. Havelund received a Ph.D. in computer science from the University of Copenhagen in 1994. Contact him at klaus.havelund@jpl.nasa.gov.

Sayan Mitra is a professor of electrical and computer engineering at the University of Illinois at Urbana–Champaign, Champaign, Illinois, USA. His research interests include the formal verification and synthesis of cyberphysical and autonomous systems. Sayan received a Ph.D. in computer science from the Massachusetts Institute of Technology. Contact him at mitras@illinois.edu.

Corina Pasareanu is the technical professional leader in data science for KBR, Houston, Texas, USA. She is part of the NASA Ames Robust Software Engineering group, performing research in software engineering and is affiliated with the Carnegie Mellon University (CMU) CyLab, CMU Silicon Valley, and the CMU Department of Electrical and Computer Engineering. Pasareanu received a Ph.D. in computer science from Kansas State University. Contact her at corina.s.pasareanu@nasa.gov or pcorina@andrew.cmu.edu.

Sanjit A. Seshia is a professor in the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, Berkeley, California, USA. His research interests include formal methods for dependable and secure computing, with application to cyberphysical systems, computer security, machine learning, and robotics. Seshia received a Ph.D. in computer science from Carnegie Mellon University. He is a Fellow of IEEE and the Association for Computing Machinery. Contact him at sseshia@eecs.berkeley.edu.

Oleg Sokolsky is a research professor in the Department of Computer and Information Science, University of Pennsylvania, Philadelphia, Pennsylvania, USA. He is member of the university's Research in Embedded Computing and Integrated Systems Center and Real-Time Systems group. His research interests include ensuring the safety of real-time and cyberphysical systems (CPSs), in addition to related areas of applying formal methods to the design and verification of CPSs, formal foundations and online monitoring for embedded systems and CPSs, hybrid systems, the automated extraction of specifications from source code, and formal methods in software engineering, particularly embedded software. He serves as *Computer*'s area editor for cyberphysical systems. Contact him at sokolsky@seas.upenn.edu.

Sanjit A. Seshia: There is a spectrum of FV methods, from assertion-based testing and model-based testing to static analysis and model checking and interactive theorem proving. So, if we define FV broadly to include this entire spectrum, I contend that FV is already used widely in software in much the same way as it is employed in hardware. Of course, software comes in many different flavors, and so we will find FV used more for software in safety-critical, mission-critical, and high-availability applications. FV is also used in certain industrial CPS applications; for example, the simulation-based falsification of temporal logic (TL) properties has been successfully applied in the automotive industry.[11] Over the past decade, I have seen big growth in interest from the CPS industry in applying formal methods to CPS design, although that interest has yet to fully translate into a wider deployment of tools.

Oleg Sokolsky: The main reason is that the software verification problem is inherently much harder. Software tends to be much less structured and much more complex compared to hardware. Finite-state models, which are much easier to verify than infinite-state ones, are a more natural fit for hardware than for software. From this perspective, CPSs are likely to make verification problems only harder. Embedded processors are becoming ever more powerful, enabling more and more complex software on board. In addition to software, physical environments need to be included in the model, making the challenge even bigger. If there is any silver lining, modern CPSs—in particular, autonomous vehicles—offer more room for lighter-weight applications of FV. Runtime verification techniques, that is, formally specified monitoring and adaptation, as well as applications of online reachability computation, appear to be very promising in autonomous CPSs.

Computer: An often neglected issue related to FV is the reliance of most techniques on expressively weak and hard-to-use formal specification languages (in the sense of creating correct specifications), such as dialects of TL. How serious do you think this problem is, and how can it be addressed?

Åkesson: For maintenance reasons, it is important to ensure that specifications and implementations are closely linked. Specifications have to be understandable by the engineer doing the implementation, and they have to be refined during the implementation phases. It should also be possible for the same engineer to update them. In our experience working with industrial partners, writing correct specifications is challenging, and it is often the case that an identified violation of a specification is due to a mistake in the formalization of the specification and not in the implementation. While FV tools have a well-defined specification language, it might be useful to consider high-level, domain-specific specification languages that integrate well with the implementation language and to consider automatically translating from this domain-specific language to the FV specification language being used.

Giannakopoulou: Creating specifications is typically an exploratory process aimed at nailing down the intended behavior of a target system, avoiding overspecification, underspecification, and ambiguity. What FV requires is a lack of ambiguity and a formal language to communicate with. In terms of ambiguity, even a simple sentence containing a condition under which some system behavior is expected has many possible interpretations. Figuring out the interpretations and picking the intended one is not straightforward. Writing a formal specification that is precise with respect to the intended interpretation is even harder. In my experience, nontrivial specifications are challenging even for experts. A way to address this problem is to build environments that assist in the process of gradually constructing specifications that are unambiguous and capture user intentions. Such environments would ideally enable users to write and explore their specifications through a variety of approaches: natural language, diagrams, use case scenarios, and interactive simulation. Formal specifications should then be produced automatically and through trusted algorithms. The problem of producing specifications can also be alleviated through the support of domain-specific specification patterns. Even in this case, however, it is crucial to provide a user-friendly environment for exploring and understanding the details of such patterns.

Havelund: Two problems are mentioned here: expressively weak specification languages and hard-to-use specification languages. I think the second problem, with hard-to-use specification languages, might be a nonissue. Just consider the complexity of C++, which programmers happily learn. Specification languages are no harder to learn, and in many cases, they are simpler than programming languages. Programmers have no difficulty writing the programs, so they can probably write specifications, as well. Some of the more simplistic languages (such as linear TL) can be hard to use for writing more complex properties, but there are solutions to that, such as specification patterns and graphical solutions, potentially translated into the harder-to-use formalisms. The real problem, in my view, might not be the difficulty of learning a specification language but the lack of willingness among developers to deal with another complex language in addition to the programming language. There is an argument for developing specifications in the programming language itself. Specification languages must be highly expressive to meet practical needs. I have developed numerous specification languages for software monitoring, and it is usually the languages that support an escape to a general-purpose programming language (when the logic formalism falls short) that appear most attractive to users.

Mitra: Verification tools must communicate with developers using artifacts and interfaces that are already part of their workflow. Requiring developers to learn a new language or a formalism is a nonstarter. Chong et al. discuss a four-year experience in which the loss of expressive power (or not using TLs, for example) was more than offset by the benefits of using the same programming language for coding and specifications.[9] This is a recurring theme at other firms adopting FV. Using common artifacts and interfaces reduces the "developer's cognitive burden and allows them to view proofs as 'just another test suite,' albeit a vastly more thorough one."[9] The integration of development and verification workflows was also a precursor to the success of hardware verification through description languages such as VHDL and Verilog.

One challenge for CPS verification is that existing tools—of which there are many strong ones—rely on mathematical models that are disconnected from developer workflows. There are no open and standard CPS languages and development ecosystems for plugging in verification tools. MATLAB is popular but, unfortunately, neither open nor standardized. The solution is to move away from model verification tools to tools that verify CPS code written in open languages, such as C, C++, and Rust, and testing and verification environments that use open simulators, including CARLA (https://carla.org/) and Gazebo (http://gazebosim.org/).

Second, some CPS components have to be treated as black boxes. The code for a component may be too complex, and it may be proprietary. The physical models may be impossible to represent as $\dot{x} = f(x)$ or as a hybrid automaton. For such black-box components, verification has to rely on statistical methods. We will need to integrate verification approaches that can combine black-box methods with model-based techniques within the development ecosystem. One approach in this direction is discussed in our DryVR framework, which has been used to verify several

ONE CHALLENGE FOR CPS VERIFICATION IS THAT EXISTING TOOLS—OF WHICH THERE ARE MANY STRONG ONES—RELY ON MATHEMATICAL MODELS THAT ARE DISCONNECTED FROM DEVELOPER WORKFLOWS.

industrial-scale systems that combine black- and white-box components.[12,13] TLs have been fundamental in understanding the complexity of verification and synthesis problems with respect to different specification classes. Extrapolating those scientific advances to a world in which developers learn TLs and start using them as specification languages for day-to-day development, in my view, is not realistic.

Pasareanu: Formal specification languages are hard to understand even for an expert in formal methods. Natural language representations, patterns, and tool support can perhaps address the problem.

Seshia: I think we can learn a lot from hardware verification. TL-based assertion languages are now widely used in hardware design, and yet the average developer does not need to be an expert in logic to use them. They have been incorporated into more accessible assertion languages, integrated with user interfaces, and generated by tools for the automated inference of specifications. In fact, tools for specification mining, learning properties from execution and simulation traces, are a very promising approach for easing the specification burden. In our own work with industry, we have seen that a specification mining tool can ease the initial burden of writing TL properties, which demonstrates to industrial users the value of formal specifications, becoming a virtuous cycle where users actively seek to learn to write logic properties due to the added value it brings them.[14] Specifications can also be integrated as "blocks" into tools that industrial users already employ; for instance, see Kapinski et al.[15]

Sokolsky: There are two related problems here. One is that, indeed, formal specification languages are hard for engineers to fully understand and use effectively. To a large extent, this drives the need for formal methods experts and stands in the way of transferring verification technology to engineers. The other problem is that, as specifications become more complex and harder to grasp, they become increasingly error prone themselves. Both issues can be partially addressed with better specification languages and tool support.

Computer: There is a perception that human involvement in the creation of formal specifications limits our ability to apply FV to CPSs. ML-based specifications are limited, at present. Can specifications created by ML algorithms be trusted? In other words, who will guard the guard (the first guard being ML-created formal specifications used for FV)?

Åkesson: Writing high-quality specifications is a very challenging task for both humans and computers. But algorithms (AI, ML, and others) can play an important role in assisting humans by proposing specifications and suggesting possible extensions. I believe that the process of formalizing specifications is as important as the verification process. During this, assumptions have to be expressed explicitly, and it has to be defined what the expected behavior should be for all corner cases. These insights are lost if ML is used to generate specifications. Thus, I see that the primary role for ML is in assisting humans by helping with the process of identifying untold assumptions and corner cases.

Giannakopoulou: Specification mining is not a new idea. In fact, several approaches have been developed that try to bypass human involvement in the creation of formal specifications. Naturally, ML is also involved in this quest. After all, it is, by now, involved in every aspect of software engineering. In my experience, ML is extremely tricky to get right, as it relies on the amount and quality of available training data and may not transfer well to other domains. One avenue that is being explored toward increasing trust is to develop ML frameworks that explain their decisions. In general, I believe we have quite a bit of work to do before we can trust ML to produce correct specifications, especially if we are liable for them. On the other hand, ML could be a valuable aid for CPS designers toward discovering, formulating, and repairing specifications.

Havelund: Specifications generated by ML techniques will undoubtedly become increasingly important. Just from a philosophical point of view, it is an evident trend. It is, however, nearly impossible to predict how much such systems can be trusted. They will, for sure, play advisory roles and eventually safety-critical ones. The most obvious approach to deal with such systems, in my view, is to monitor their execution and ensure that they behave within a more traditionally defined safety region. Hence, the guards of the guards are monitors. ML can also be used to propose formal specifications to be approved by humans.

Mitra: When specification writing becomes part of the development process, with tangible benefits, and it is no longer seen as an isolated activity, then the creation of specifications may not be viewed as burdensome. One study reports that AWS developers spend considerable energy writing proof harnesses, which are essentially assertions written in the programming language and that guide the verification engine and provide much better coverage.[9] ML-created specifications are an intriguing idea. Obviously, generating labeled data for any such approach will still require curation and expertise.

Pasareanu: I believe there will always be some human involvement and domain expertise in the creation of formal specifications. I am not sure what you have in mind with "ML-based specifications." If these are specifications mined from data and/or systems, then I think a human expert can validate them. FV tools can be used to formally verify them.

Seshia: As I mentioned earlier, learning specifications from data and other artifacts is a promising approach to ease the specification burden. One way to generate trust in ML-created formal specifications is to validate them against available code and models, with human oversight. This is exactly the approach we took in a collaboration with Toyota, where, when an engineer felt our generated specification was incorrect, the validation pointed to a corner case bug in a large Simulink model the company was analyzing.[14] In other words, mining specifications and finding corner case bugs are two sides of the same coin. This specification mining approach is a special case of a more general methodology for high-assurance ML termed *oracle-guided learning* or *oracle-guided inductive synthesis*.[16,17]

Sokolsky: On the one hand, we clearly need a way to keep tabs on machine-generated specifications, to make sure they capture our intuitive goals and that there are no unintended aspects. On the other hand, we must remember that human-created specifications are not perfect, either. Thus, the question is not whether we should trust machine-generated specifications more or less than ones crafted by humans. Whatever the source, we should be able to perform sanity checks on a specification or, better yet, verify it with respect to higher-level requirements.

Computer: How much of the verification of a CPS is physics, and how much is logic and traditional reasoning tools? Where do you think this ratio is headed? Similarly, how much is logical inference versus statistical inference? How much of ML algorithms can translate into traditional reasoning, and what is lost in the process?

Åkesson: Physics plays a vital role in restricting the behavior of a closed-loop system. However, it is the perception and decision-making code that is rapidly increasing in complexity.

Giannakopoulou: Instead of commenting on the ratio of physics to logic, I will share some observations. In my experience, many novel algorithms for autonomous decision making (collision avoidance, for example) are constructed using models (often probabilistic) of the physical systems involved. Finding the right level of model abstraction to combine scalability with safety is an art. Ensuring the conformance of physical models to the real world is key when verifying CPSs. The need to deal with uncertainty and optimization, which are intrinsic in autonomy, creates a natural shift toward statistical inference. In my opinion, the major challenge with reasoning about ML algorithms is that their logic is not explicit, making it hard to formulate and assess the correctness of their behavior.

Havelund: As long as there is traditional software in CPSs, it will need to be verified and tested. Furthermore, such systems will increase in complexity, meaning even more software to be verified and tested. A big part of such future systems will therefore be traditional testing and logic-based reasoning tools to the extent that they scale to the problem.

Mitra: The physics-to-logic ratio in CPS verification evolves across development stages. As physical processes become better understood and controlled, design and verification complexity shifts to the computing stack, with the goals of achieving better efficiency, less energy use, and utilization. The early adopted methods are usually the ones that are stable and easier to interpret. My view is that the early adoption of CPS verification will be dominated by the more traditional proofs, logical inference, and absolute guarantees, while statistical approaches will dominate testing. For end-to-end and system-level verification, the verification results of heterogenous components have to be composed. There are very interesting ideas about incorporating ML in verification, particularly for handling black-box components we mentioned earlier, but these approaches are still in their infancy.

Pasareanu: I think it is hard to quantify. It seems, indeed, that we have a bit of all of them.

Seshia: Your first question goes to the crux of how CPSs are defined. According to Edward Lee and myself, CPSs are integrations of computation with physical processes whose behavior is defined by both cyber and physical components.[18] Thus, every CPS verification problem involves reasoning about the "physics" and reasoning about computation. Now, to achieve scalability, we typically must take a modular approach, where we break up the CPS verification problem into several subproblems, some purely cyber, some purely physical, and some cyberphysical. With respect to your second question, I think inductive learning, also known as *ML*, is central to the process of proof. The combination of inductive and deductive reasoning has been at the heart of many advances in FV over the past 20 years, including counterexample-guided abstraction refinement and techniques for invariant synthesis, where inductive learning is combined with deductive reasoning by using hypotheses about the structure of proof artifacts being synthesized.[16] So, ML algorithms do fit in a natural way into "traditional" reasoning. It remains to be seen how useful deep learning, specifically, will be in FV.

INTEGRATION AS AN AFTERTHOUGHT USUALLY RESULTS IN EXPENSIVE REDESIGNS AND MODIFICATIONS LATE IN THE SOFTWARE DEVELOPMENT LIFE CYCLE.

Sokolsky: It seems hard to separate the effects of physics and logic in CPS verification challenges. While physics verification seems harder, or at least less scalable than logic verification, it is the interaction between physics and logic that makes CPS verification so difficult. The balance between logical and statistical inference depends on the verification approach, with statistical inference becoming ever more prominent in recent years.

Computer: Should the FV of a CPS be conducted on the interface between the cyber and physical partitions instead of directly on them?

Åkesson: There is a need to do both. During early development phases, the components and their interfaces are defined, and the implementation and models might be missing or incomplete. During these phases, the interfaces' expectations and guarantees toward the environment can be defined and verified. Later in the development process, FV and other rigorous test methods, such as falsification, should be used to verify the closed-loop behavior.

Giannakopoulou: This falls under the standard topic of unit versus integration testing/verification. The answer is that it should be conducted at all levels. However, given the complexity of CPSs, it is worthwhile to invest in studying the interface between the cyber and physical partitions first. Understanding and specifying the intended interactions between the two provides a solid foundation for developing systems that will integrate seamlessly. Integration as an afterthought usually results in expensive redesigns and modifications late in the software development life cycle.

Havelund: As I pointed out, I think a large part of the verification of CPSs will still be the validation of traditional code bases. However, specifically monitoring techniques, also referred to as *runtime verification*, can be used to oversee the interface between the software and the physical system and potentially prevent the software from doing any harm, a subfield of runtime verification referred to as *runtime enforcement*. Here, the monitor will prevent the software from issuing harmful commands to the physical system.

Mitra: Carefully defining CPS model interfaces can help achieve a separation of concerns, for example, farming out the physics models or components and the software elements to different proof engines in such a way that their results can be soundly combined to verify the overall model. Our Koord[19] language and the CyPhyHouse[20] verification framework are tailored to address this issue in the context of distributed CPSs written using shared memory.

Pasareanu: Perhaps on both. Compositional reasoning can be helpful in putting together results from separate verifications.

Seshia: Since CPSs are fundamentally about the intersection between cyber and physical worlds, some verification will always need to be on the interface between the two. For compositional analysis, some verification may need to be on individual cyber and physical "partitions." But the overall proof will always involve the interface. And if a counterexample is to be demonstrated, it must be a full CPS counterexample. Our experience working with industrial users in the automotive sector is that, first, integration testing is the biggest challenge, and second, people care much more about system-level counterexamples than "unit" counterexamples; for instance, see the work of Yamaguchi et al.[11]

Sokolsky: I try to avoid being prescriptive in the choice of verification approaches. Whatever works should be used. I would imagine that interface-based techniques may offer better scalability, in general, at the expense of more significant conservatism. A lot depends on the system design, and the verification engineer should be prepared to apply the whole range of available tools as needed.

Computer: Of the current impediments—technical or otherwise—that make it challenging to formally verify a CPS in an effective and efficient manner, which do you think is the most pressing to address and why?

Åkesson: Scalability and ease of use are limiting the industrial acceptance of the FV of CPSs. The limitations of formal and rigorous verification methods signify the importance of a modular approach, such as combining ML components with correct-by-construction approaches and software modules with manageable complexity. A significant challenge is combining the white-box approaches of FV with the black-box methods used in falsification to handle systems where parts are fully known while for others, only incomplete information is available.

Giannakopoulou: Regulatory bodies are pressed to come up with solutions for ensuring the safety of autonomous vehicles, which are expected to invade our lives in massive numbers in the near future. It is a great opportunity to exploit this pull for techniques that ensure trust in autonomy. In many respects, CPSs share verification challenges with traditional large, complex distributed systems and can benefit from advances made in those domains. However, they place increased emphasis on AI. Within that domain, I believe it is most pressing to identify and formulate requirements for the correctness of adaptive and ML algorithms.

Havelund: The main problem, in my view, is the algorithmic challenge in verifying large systems. We are currently not able to automate this process sufficiently to make it broadly attractive. To this can be added the problem of writing specifications. However, I do believe that if the verification problem could be solved (highly automated) and if specifications really captured the details of interest (requiring expressive specification languages), there could be enough motivation for adopting FV. This is not to underestimate the problem of writing specifications. There is a need to support the formal specification and verification of programs written in programming languages and perhaps with specifications written in the programming language itself, for example, much like unit tests. Some programming languages are now being developed with built-in support for FV. The guaranteed short-term-winner approach is automated testing 24/7, in which a system is constantly bombarded with inputs and monitored as it executes with advanced test oracles. This requires trustworthy simulators of the physical systems, which can be rerun repeatedly on a normal desktop or laptop.

REGULATORY BODIES ARE PRESSED TO COME UP WITH SOLUTIONS FOR ENSURING THE SAFETY OF AUTONOMOUS VEHICLES, WHICH ARE EXPECTED TO INVADE OUR LIVES IN MASSIVE NUMBERS.

Mitra: We need a standardized, open development ecosystem for CPSs and related benchmarks. Open standards help identify problem definitions and attract talented researchers. They reduce friction in sharing solutions. Benchmarks and standards also help practitioners share hard instances across domains, and they give a yardstick for the community to measure progress.

Pasareanu: CPSs are increasingly built using ML components, such as neural networks, which are hard to specify and verify formally. I view that as the main challenge.

Seshia: In a sense, the CPS verification challenge is the union of the difficulties of verifying hardware, software, and physical systems because CPSs integrate all of them. It is difficult to identify a single challenge that is the "most pressing." My top contenders include modeling the complex environments of CPSs, developing better theories of compositional reasoning for CPSs, verifying intelligent CPSs based on AI and ML, and creating a large and diverse repository of benchmarks to guide the community.

Sokolsky: A lot of challenges to FV, such as the computational complexity of verification algorithms and the rapidly growing scale of CPSs, are fundamental and thus cannot be really addressed, in my opinion. What can be addressed is the verifiability of CPSs. Systems can and should be designed in a way that makes them easier to verify, more modular, and better structured. To achieve that, we need better design approaches and techniques. But even more importantly, we need to change the mindset of designers. Most system designers are not experts in formal methods and do not need to be. But they need a better understanding, if only at a rule-of-thumb level, of what makes a system easier or harder to verify.

There is consensus among the panelists that the software industry is, indeed, slow to adopt FV, except for static analysis—which is arguably more of a compiler technology than FV—and some projects run by deep-pocket companies. The reasons include software's complexity, rate of change, and diverse correctness criteria. A key obstacle cited by multiple experts is the FV environment and ease of use. In contrast with the EDA market, in which FV is a first-class member of the development environment and tool chain, for software developers, FV is like a distant "nerd" cousin that speaks a different dialect and one that few first-class members pay attention to.

The panel members agree that as difficult as it is to successfully apply FV to software in general, it is as difficult or more so to apply it to CPSs. Some argue that the complexity, scale, and opaque nature of ML algorithms make the full application of FV for CPSs unrealistic. However, limited approaches, such as runtime verification (especially on the interface between the cyber and physical partitions of a CPS) can be used. The expectation that, inevitably, some correctness properties themselves will be machine learned only exacerbates the trust problem. Nevertheless, despite such mounting challenges, we recommend that the FV research community measure up by developing techniques for dealing with the difficult nature of building dependable CPSs. As an analogy, consider the various techniques theoretical computer scientists have developed for coping with intractable (NP-complete) problems, such as heuristics, Horn logic, and Boolean satisfiability-solving algorithms. Indeed, runtime verification is one such approach.

DISCLAIMER

The views and conclusions contained herein are those of the panelists and moderators and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of their employers. The U.S. government is authorized to reproduce and distribute reprints for government purposes, notwithstanding any copyright annotations thereon.

REFERENCES

1. M. B. W. Tent, ed., *Gottfried Wilhelm Leibniz: The Polymath Who Brought Us Calculus*, 1st ed. Boca Raton, FL: CRC Press, 2011.
2. E. W. Dijkstra, "On the reliability of mechanisms," in *Notes on Structured Programming*, T. H.-Report 70-WSK-03, 2nd ed. Eindhoven, The Netherlands: ersity, Apr. 1970, p. 7. Accessed: June 1, 2021. [Online]. Available: https://www.cs.utexas.edu/users/EWD/ewd02xx/EWD249.PDF
3. A. Edelman, "The mathematics of the Pentium division bug," *SIAM Rev.*, vol. 39, no. 1, pp. 54–67, 1997. doi: 10.1137/S0036144595293959
4. "*Intel takes $475-million earnings hit: Computers: The charge for replacing flawed Pentium chips mars an otherwise stellar year*," Los Angeles Times, Jan. 18,

1995. Available: https://www.latimes.com/archives/la-xpm-1995-01-18-fi-21424-story.html

5. *IEEE Standard for SystemVerilog—Unified Hardware Design, Specification, and Verification Language*, IEEE Standard 1800-2017 (Revision of IEEE Standard 1800-2012), Feb. 22, 2018. doi: 10.1109/IEEESTD.2018.8299595.
6. J. B. Michael, G. W. Dinolt, and D. Drusinsky, “Open questions in formal methods,” *Computer*, vol. 53, no. 5, pp. 81–84, 2020. doi: 10.1109/MC.2020.2978567.
7. “Letters: Another view on formal methods,” *Computer*, vol. 53, no. 9, p. 8, 2020. doi: 10.1109/MC.2020.3001958.
8. C. Sadowski, E. Aftandilian, A. Eagle, L. Miller-Cushon, and C. Jaspa, “Lessons from building static analysis tools at Google,” *Commun. ACM*, vol. 61, no. 4, pp. 58–66, 2018. doi: 10.1145/3188720.
9. N. Chong et al., “Code-level model checking in the software development workflow,” in *Proc. ACM/IEEE 42nd Int. Conf. Softw Eng.: Softw. Eng. Pract.*, 2020, pp. 11–20. doi: 10.1145/3377813.3381347.
10. P. W. O’Hearn, “Continuous reasoning: Scaling the impact of formal methods,” in *Proc. 33rd Annu. ACM/IEEE Symp. Logic Comput Sci.*, 2018, pp. 13–25. doi: 10.1145/3209108.3209109.
11. T. Yamaguchi, T. Kaga, A. Donze, and S. A. Seshia, “Combining requirement mining, software model checking, and simulation-based verification for industrial automotive systems,” in *Proc. IEEE Int. Conf. on Formal Methods Computer-Aided Design*, Oct. 2016, pp. 201–204. doi: 10.1109/FMCAD.2016.7886680.
12. C. Fan, B. Qi, S. Mitra, and M. Viswanathan, “DryVR: Data-driven verification and compositional reasoning for automotive systems,” in *Computer Aided Verification* (Lecture Notes in Computer Science), R. Majumdar and V. Kunčak, Eds. Berlin: Springer, pp. 441–461. 2017. doi: 10.1007/978-3-319-63387-9_22.
13. S. Mitra, *Verifying Cyber-Physical Systems: A Path to Safe Autonomy*. Cambridge, MA: The MIT Press, 2021. ISBN-13: 978-0262044806.
14. X. Jin, A. Donze, J. Deshmukh, and S. A. Seshi, “Mining requirements from closed-loop control models,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 34, no. 11, pp. 1704–1717, 2015. doi: 10.1109/TCAD.2015.2421907.
15. J. Kapinski et al., “*ST-Lib: A library for specifying and classifying model behaviors*,” SAE Tech. Paper 2016-01-0621, 2016. doi: 10.4271/2016-01-0621.
16. S. A. Seshia, “Combining induction, deduction, and structure for verification and synthesis,” *Proc. IEEE*, vol. 103, no. 11, pp. 2036–2051, 2015. doi: 10.1109/JPROC.2015.2471838.
17. S. A. Seshia, D. Sadigh, and S. S. Sastry, “Towards verified artificial intelligence,” July 2016. [Online]. Available: https://arxiv.org/abs/1606.08514
18. E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*, 2nd ed. Cambridge, MA: The MIT Press, 2017.
19. R. Ghosh, C. Hsieh, S. Misailovic, and S. Mitra, “Koord: A language for programming and verifying distributed robotics application,” *Proc. ACM Program. Lang.*, vol. 4, pp. 1–30, 2020. doi: 10.1145/3428300.
20. R. Ghosh et al., “CyPhyHouse: A programming, simulation, and deployment toolchain for heterogeneous distributed coordination,” in *Proc. IEEE Int. Conf. Robotics Automat.*, 2020, pp. 6654–6660. doi: 10.1109/ICRA40945.2020.9196513.

JAMES BRET MICHAEL is a professor in the Department of Computer Science and the Department of Electrical and Computer Engineering, Naval Postgraduate School, Monterey, California, 93943, USA. Contact him at bmichael@nps.edu.

DORON DRUSINSKY is a professor in the Department of Computer Science, Naval Postgraduate School, Monterey, California, 93943, USA, and the chief science officer at Aerendir, Mountain View, California, 94040, USA. Contact him at ddrusins@nps.edu.

DUMINDA WIJESEKERA is a professor in the Department of Cyber Security Engineering, George Mason University, Fairfax, Virginia, 22030, USA, where he is codirector of the Center for Assured Research. Contact him at dwijesek@gmu.edu.

EDITOR: Robert Blumen, SalesForce, robert@robertblumen.com

This article originally appeared in IEEE Software vol. 37, no. 2, 2020

Boris Cherny on TypeScript

Nate Black

FROM THE EDITOR

In Episode 384 of "Software Engineering Radio," Boris Cherny, author of *Programming TypeScript*, speaks with Nate Black, explaining how TypeScript can scale JavaScript projects to larger teams, larger code bases, and across devices. TypeScript is a gradually typed language, allowing you to add compile-time verification to a JavaScript project bit by bit. TypeScript aims to be practical by catching common mistakes but without adding too much burden on the programmer. Other topics include: structural typing, type refinement and programmer intuition, when to use escape hatches and how to ban them, interoperability with JavaScript, and using TypeScript with frameworks such as Angular, React, and React Native. We provide summary excerpts below; to hear the full interview, visit http://www.se-radio.net or access our archives via RSS at http://feeds.feedburner.com/se-radio.—*Robert Blumen*

Nate Black: What is TypeScript?

Boris Cherny: TypeScript is a programming language that compiles to JavaScript. It mostly came out of Microsoft, with contributions from Google. Its type system is structural, static, strong, mostly inferred, and gradually typed. TypeScript is the most successful among gradually typed languages and the most popular language that compiles to JavaScript.

Why are there so many languages that compile to JavaScript?

JavaScript can run on every computer and every phone. Because it's so ubiquitous, Facebook, Google, and others have built big applications (apps) on it. But it lacks fundamental features, such as static types that are necessary to scale a program across more engineers and more devices Compile-to-JavaScript languages are intended to overcome limits to scalability.

How similar or dissimilar is TypeScript from JavaScript?

TypeScript compiles to and interoperates well with JavaScript. Every valid JavaScript program is also a TypeScript program. It might not type check, but it will compile. If you have a JavaScript file that ends with a .js and rename it to a .ts extension, that's a TypeScript program.

What does it mean to scale?

It means more lines of code, more engineers, and more devices. Big tech companies have millions of lines of code. Code will start breaking when you make changes to it if the consequences of changes are not

Digital Object Identifier 10.1109/MS.2019.2958155
Date of current version: 12 February 2020

totally clear. Static type has helped solve that problem; when you modify a line of code or the app programming interface to some function or some kind of module, you know exactly what else it will break for certain classes of errors.

TypeScript scales across more engineers, serving as documentation when type notations are added to functions (for example, "Function F accepts a number and returns a string"). It scales across more devices; once you start scaling to services and writing multithreaded JavaScript, TypeScript enforces well-defined protocols by letting you type both sides of the communication, thereby increasing confidence. For example, you can write TypeScript that runs on both the browser and server.

How does the compilation process work when JavaScript is run through the TypeScript compiler? What does the TypeScript compiler do?

JavaScript is an interpreted language. When you put JavaScript code into a text file, feed it into your browser, and then run it, it must be compiled to byte code or machine code before the user actually executes the program. It's the same with TypeScript, a language that targets JavaScript so it can run on any platform that supports JavaScript. TypeScript takes code and then outputs JavaScript code, which you can then run as you would have before.

You wrote that changing type definitions of the TypeScript program won't change the compiled JavaScript output. How is that possible and what does that mean?

TypeScript is JavaScript plus some types. The types will never affect the output of the program, so you can do what you want with the types. You can make it safer or less safe by using different kinds of types. But the generated JavaScript output will look exactly the same. The types affect type checking done by the TypeScript compiler before it compiles. But it can still compile your code even if it doesn't type check or if the types don't totally work. If you want to opt into more safety, TypeScript has various safety flags, one of which controls this behavior and will not admit code unless the program type checks.

SOFTWARE ENGINEERING RADIO

Visit www.se-radio.net to listen to these and other insightful hour-long podcasts.

RECENT EPISODES

- » 391—Jeremy Howard from fast.ai explains deep learning from concept to implementation. He and host Nate Black discuss neural network architecture and deep-learning models.
- » 390—Sam Procter, a researcher at the Software Engineering Institute of Carnegie Mellon University, discusses security in software design with host Justin Beyer.
- » 387—Host Gavin Henry speaks with Abhinav Asthana, a founding partner and chief executive officer of the API development tool Postman, about API design and testing, where to start, which types of APIs to offer, what tools you can use, what features to expose, and which is his favorite API to reference.

UPCOMING EPISODES

- » Host Robert Blumen talks with Jay Kreps on event-based log architecture with Kafka.
- » Chris McCord discusses the Phoenix framework with host Adam Conrad.
- » Pat Helland speaks with host Akshay Manchale about data management at scale.

How can it compile the code even if it doesn't type check?

TypeScript looks at the value level of your program as opposed to the type level—the JavaScript part of it, not the TypeScript part—to compile it. The types are used just for type checking, before it compiles.

If the type checking conveys information that it didn't type check correctly, how would I respond?

When you write TypeScript in an editor that supports it, you will get errors in your text editor warning you that you made a mistake. You can configure your project such that unless there are no errors, the code doesn't compile. Type errors alert you to likely mistakes in your program that you should probably fix, but if you want, you can ignore them.

What is the paradigm behind TypeScript?

The types are there when you need them but not when you don't. TypeScript supports various paradigms of programming: functional, object oriented, and imperative. TypeScript can infer all the types for you, or you can specify them explicitly. It works well with all these styles of programming. The idea is that types are fun and they're useful, so it shouldn't be tedious to fix them.

What more can you say about interoperability between TypeScript and JavaScript?

Modern JavaScript programs use a lot of code. JavaScript is a very modular language, so you often have first-party JavaScript code along with stuff from third-party packages that might be written in TypeScript. Your code might be a mix of JavaScript and TypeScript. This is a use case that TypeScript had to support when it was designed. It's important in JavaScript to be able to use whatever packages you want and for it to work correctly. If these are written in TypeScript, you also get the benefit of type checking and autocompletion in your text editor for free.

Can you isolate those parts of the code that have less strictness or where you don't have the type information available?

Because TypeScript is built to be practical, you might want to interoperate with JavaScript code. A common pattern is to start with JavaScript code and then gradually migrate the code base to TypeScript. To migrate from JavaScript to TypeScript one piece at a time, you just rename your file type with a .ts extension. You can still use JavaScript and opt that part of the code base into the strong safety guarantees. Over time as you migrate more and more of your code, you can flip on these safety settings one by one.

I've read that most runtime errors in JavaScript are type errors. Was the goal for TypeScript to eliminate that class of errors?

Yes. Some mistakes are harder for a programmer to identify than others. Mistakes might stem from architectural issues that will take a design review to walk through and understand. TypeScript makes it easy to prevent the really dumb mistakes that could have been caught but weren't because you weren't using types.

NATE BLACK is a software engineer at Sleeperbot, an online sports community. Contact him at nathanael.black@gmail.com.

PURPOSE: The IEEE Computer Society is the world's largest association of computing professionals and is the leading provider of technical information in the field.

MEMBERSHIP: Members receive the monthly magazine *Computer*, discounts, and opportunities to serve (all activities are led by volunteer members). Membership is open to all IEEE members, affiliate society members, and others interested in the computer field.

COMPUTER SOCIETY WEBSITE: www.computer.org

OMBUDSMAN: Direct unresolved complaints to ombudsman@computer.org.

CHAPTERS: Regular and student chapters worldwide provide the opportunity to interact with colleagues, hear technical experts, and serve the local professional community.

AVAILABLE INFORMATION: To check membership status, report an address change, or obtain more information on any of the following, email Customer Service at help@computer.org or call +1 714 821 8380 (international) or our toll-free number, +1 800 272 6657 (US):

- Membership applications
- Publications catalog
- Draft standards and order forms
- Technical committee list
- Technical committee application
- Chapter start-up procedures
- Student scholarship information
- Volunteer leaders/staff directory
- IEEE senior member grade application (requires 10 years practice and significant performance in five of those 10)

PUBLICATIONS AND ACTIVITIES

***Computer*:** The flagship publication of the IEEE Computer Society, *Computer* publishes peer-reviewed technical content that covers all aspects of computer science, computer engineering, technology, and applications.

Periodicals: The society publishes 12 magazines and 17 journals. Refer to membership application or request information as noted above.

Conference Proceedings & Books: Conference Publishing Services publishes more than 275 titles every year.

Standards Working Groups: More than 150 groups produce IEEE standards used throughout the world.

Technical Committees: TCs provide professional interaction in more than 30 technical areas and directly influence computer engineering conferences and publications.

Conferences/Education: The society holds about 200 conferences each year and sponsors many educational activities, including computing science accreditation.

Certifications: The society offers three software developer credentials. For more information, visit www.computer.org/certification.

BOARD OF GOVERNORS MEETING

1-3 February 2022

revised 16 December 2021

COMPUTER SOCIETY OFFICES

Washington, D.C.: 2001 L St., Ste. 700, Washington, D.C. 20036-4928; **Phone:** +1 202 371 0101; **Fax:** +1 202 728 9614; **Email:** help@computer.org
Los Alamitos: 10662 Los Vaqueros Cir., Los Alamitos, CA 90720; **Phone:** +1 714 821 8380; **Email:** help@computer.org

MEMBERSHIP & PUBLICATION ORDERS

Phone: +1 800 678 4333; Fax: +1 714 821 4641; Email: help@computer.org

This article originally appeared in IEEE SECURITY & PRIVACY vol. 19, no. 2, 2021

Toward Unseating the Unsafe C Programming Language

Paul C. van Oorschot, *Associate Editor in Chief*

Reflecting on content that I taught in a recent security course about software-based vulnerabilities, I wondered: Am I giving too much focus to the C programming language? C-based examples get straight to the point, allowing compact illustrations of the concepts underlying stack- and heap-based buffer overruns and return-oriented programming, aside from integer-based vulnerabilities, related to arithmetic underflow, conversions between signed and unsigned values, and errors due to the compiler promotion of short-integer data types in arithmetic expressions.[1] But are these relevant for today's students, given the wide availability of modern languages with strong language safety properties? Unfortunately, the answer is still yes.

C retains a stubborn hold as the dominant systems-level language, despite its longstanding issues. Its pointers allow direct memory references (that is, by explicit address), and it allows pointer arithmetic and programmatic conversion between data types (for example, casting integers to pointers) while lacking language-based enforcement of array bounds. Programmers are responsible for checking that memory references access addresses consistent with the declared data structures. They are also responsible for dynamic memory allocation and release. In C, neither compile-time nor run-time checks prevent a variable of one type being changed to, or interpreted as, another type; thus, there are no guarantees on the kinds of data that a given variable might hold, the allowed set of values, their representation, or the operations that may be carried out on the data. Nonetheless, we continue to use C, politely excusing these issues by acknowledging, with benign terminology, that C is *not strongly typed*.[2] Less euphemistically, because arbitrary integers may be repurposed as pointers, which, when dereferenced may access any memory in the address space of a process, C is said to lack *memory safety*—a handy term,[3] but one for which it is hard to give a definition that is both precise and useful toward resolving related problems. Moreover, the problems themselves are both well understood by experts and have been experienced by all serious C programmers. The challenges and related unlikelihood of ever replacing all legacy C code is one thing. However, having learned our lesson from 45 years of use, surely we do not still use C in new projects and in building brand new systems, do we?

As it turns out, the evidence suggests we do. In background reading, I came across a comprehensive survey by Hahm et al.[4] aiming to identify top-candidate operating systems (OSs) as a platform for constrained devices in the Internet of Things (IoT). Beyond the discussion of commercial OSs, such as QNX and ARM's Mbed OS, a summary table highlights 12 open source IoT OSs, including RIOT, Contiki, FreeRTOS, and TinyOS. Here is what caught my eye: all 12 are C based (TinyOS is based on nesC, a C variant; RIOT is based on C and C++). Can this really be true—in a 2016 article, the top 12 open source platform candidates for emerging IoT devices are all still based on C? I found this both surprising and alarming when thinking about the future problems that this presumably locks us into—problems already recognized more than 30 years ago in the aftermath of early computer worm incidents. Even back then, attention was drawn to

Digital Object Identifier 10.1109/MSEC.2020.3048766
Date of current version: 15 March 2021

the looseness of the C language. Problems were clear from the language in Kernighan and Ritchie's de facto specification, for example, as they note in a section about pointers and integers: "Certain other conversions involving pointers are permitted, but have implementation-dependent aspects."[5] In other words, the expected behavior is undefined—a red-flag word to the ears of every security expert.

Unfortunately, even if we ignore the vast quantities of legacy code written in C and consider only new projects, finding an alternative to C is not as simple as "Use type-safe languages like Java." To implement OSs, systems programmers require efficient execution, the ability to directly reference memory addresses (for example, for hardware device drivers), and, in the case of real-time OSs, worst-case execution time guarantees. In type-safe languages, such as Java, programs access objects through variable names (references) while explicit memory addresses (pointers to the objects) remain programmatically inaccessible. Dynamic memory management, in particular the allocation of regions of specifically sized heap memory and the release of such memory, is no longer a programmer's responsibility—but, as a consequence, run-time *garbage collection* is required, at the cost of losing efficiency and worst-case time guarantees.

The challenge is to find type-safe languages that also meet the requirements of systems programming. Java was never intended as a systems-level language nor as an alternative to C (Microsoft's object-oriented C# similarly targets applications more than, for example, OS kernels). Also, to be clear, Java has had its own rich history of security problems6—in part due to its global adoption making it an enormous target and, in part, due to a complex architecture with many moving pieces, including a bytecode verifier for its runtime virtual machine. Java has its own vast set of security problems despite having had both an early design focus on security as required by the initial use case of Java applets as (untrusted) downloaded active content[7] and an early major redesign based on the principles of fine-grained access control and least privilege.[8] Moreover, while a type-safe language prevents arbitrary programmatic access to memory, this does not by itself solve all errors that result in security vulnerabilities (for example, the Java language itself prevents neither integer underflows nor SQL exploits induced by unsanitized input).

But why does the previous list of 12 candidates for IoT OSs include no safe languages? One excuse is that it takes quite some time, even for promising new tools and OSs, to gain a following. Another is that the path of least resistance for developers is to continue using well-established tools due to advantages in cost, familiarity, interoperability, and backwards compatibility. How, then, do we break from the past and move beyond the dominance of C for system-level programming?

The answer may be OSs built on the Rust programming language[9] (I discuss one later, Tock; alternatively, I could have chosen the more-recent Redleaf).[10] Rust is a systems programming language with features resembling C++. Following its 2010 announcement as an open source project within Mozilla, and a Rust 1.0 release on 15 May 2015, the Rust language and compiler have gained popularity (albeit considerably more in opinion than actual use, perhaps due to anecdotal reports of a steep learning curve). Notably, Rust has attracted use by major players, including Microsoft and Amazon Web Services. From a security perspective, major features that make Rust interesting are its suitability as a systems programming language supporting direct-memory references and its memory management offering type safety (implying memory safety) without runtime garbage collection. This sidesteps timing uncertainties that are incompatible with the requirements of many OSs.

The Tock OS[11] offers a new design as a high-performance, security-focused OS for constrained platforms. It relies on the principle of minimizing the trusted computing base. A qualification on Tock's type safety is that a subset of the Tock kernel must be trusted, in the following sense: Small parts of Tock involve code that violates strict type-safety rules. The Rust language allows this by permitting designated code blocks to perform unsafe operations, for example, dereferencing untyped pointers as customary for hardware peripheral interfaces. However, full type-safety rules remain in place for applications running on top of Tock. Among other contributions in Tock's design is a mechanism for partitioning kernel-owned heap memory among active processes. Supported by Rust's design, Tock avoids dynamic memory allocation (and garbage collection), supports multiprogramming, and isolates processes from each other (and the kernel). Tock does not rely on a hardware memory-management unit, which, in traditional CPU-based systems, separates the address spaces of processes and maps the virtual address space of each process onto physical memory. It does, however, rely on some hardware features, for example, in the case of ARM processors, their memory protection unit.

A Tock kernel built for ARM Cortex-M microcontrollers demonstrates a surprisingly small footprint for the OS,[11] as given by the details for Atmel's SAM4L Cortex-M4 [48-MHz clock, 512-kB flash for code, 64-kB static RAM (SRAM)]. For this platform, Tock's core kernel was written in Rust, with custom hardware adaptations for the SAM4L platform, and a few hundred lines of assembly for context-switching code. The kernel uses 8.4-kB SRAM plus 4 kB for the kernel stack, plus 87 kB of flash for kernel code. A case study in the same article details how Tock can serve as the OS for a USB embedded device (security key), hosting several security applications written by independent developers, with OS-provided process isolation.

For broader context, numerous previous initiatives have aimed to replace C, offering better safety features while retaining its efficiency; others have proposed memory-safe C dialects, such as Cyclone.[12] None have gained a following sufficient to overcome C's dominance; incumbents enjoy natural advantages in complex software systems. Other major languages in use but not on the path to replace C for building OS kernels include Apple's Swift (itself a replacement for Objective-C, albeit with ongoing ties to C libraries) and Google's Go (with attention to efficient garbage collection and independent of C as of 2016's Go 1.5 release).

The question that we now return to is: Will the growth of the IoT and the requirement for constrained OSs suitable for low-end devices result in the ongoing proliferation of C-based OSs—delivering another 40 years of systems dominated by C? Or will security and safety issues drive us toward tools and platforms (for example, building on Rust) that promise greater security? These are, I believe, important questions for not just the security and systems software communities but also for the population of the entire world as consumers and users of cascades of IoT products.

What power do governments have to influence these decisions—or should they even get involved? The high-tech industry, particularly the software industry, has always been strongly opposed to government regulation and intervention—while the security track record of IoT manufacturers to date suggests that a "let us regulate ourselves" industry position has failed badly in terms of delivering secure devices.

One potential approach would be for major governments to use their enormous purchasing power as consumers of IT products to influence the toolsets and platforms that device manufacturers use and build on as their foundations. The well-established idea is that once manufacturers have built conformant products for purchase by major governments, they will then have and use them as a natural baseline for their full product lines, for reasons related to economies of scale. Perhaps memory-safe platforms, whether specifically involving Rust or not, will eventually make their way into requirements documents for major government purchases. In any case, the challenge is to find some way to build tomorrow's high-tech world on a software platform less vulnerable than C-based OSs. I ask you to consider: What can you do to help?

REFERENCES

1. P. C. van Oorschot, "Software security—Exploits and privilege escalation," in *Computer Security and the Internet: Tools and Jewels:* Springer-Verlag, 2020, ch. 6, pp. 155–182.
2. L. Cardelli and P. Wegner, "On understanding types, data abstractions, and polymorphism," *ACM Comput. Surv.*, vol. 17, no. 4, pp. 471–522, Dec. 1985. doi: 10.1145/6041.6042.
3. V. van der Veen, N. dutt-Sharma, L. Cavallaro, and H. Bos, "Memory errors: The past, the present, and the future," in *Proc. Int. Workshop Recent Adv. Intrusion Detection*, 2012, pp. 86–106. doi: 10.1007/978-3-642-33338-5_5.
4. O. Hahm, E. Baccelli, H. Petersen, and N. Tsiftes, "Operating systems for low-end devices in the Internet of Things: A survey," *Internet Things J.*, vol. 5, no. 3, pp. 720–734, 2016.
5. B. Kernighan and D. Ritchie, *The C Programming Language*, 2nd ed. Englewood Cliffs, NJ: Prentice Hall, 1988; 1st ed., 1978, p. 198.
6. P. Holzinger, S. Triller, A. Bartel, and E. Bodden,

"An in-depth study of more than ten years of Java exploitation," in *Proc. 2016 ACM SIGSAC Conf. Comput. Commun. Security*, pp. 779–790. doi: 10.1145/2976749.2978361.

7. G. McGraw and E. W. Felten, *Securing Java: Getting Down to Business with Mobile Code*. New York: Wiley, 1999.
8. L. Gong, "Java security architecture revisited," *Commun. ACM*, vol. 54, no. 11, pp. 48–52, Nov. 2011. doi: 10.1145/2018396.2018411.
9. S. Klabnik and C. Nichols, *The Rust Programming Language (Covers Rust 2018)*. San Francisco: No Starch Press, Aug. 2019. [Online]. Available: https://doc.rust-lang.org/book/
10. V. Narayanan et al., "RedLeaf: Isolation and communication in a safe operating system," in *Proc. USENIX OSDI*, 2020, pp. 21–39.
11. A. Levy et al., "Multiprogramming a 64 kB computer safely and efficiently," in *Proc. 26th Symp. Oper. Syst. Principles*, 2017, pp. 234–251. doi: 10.1145/3132747.3132786.
12. T. Jim, J.G. Morrisett, D. Grossman, M.W. Hicks, J. Cheney, and Y. Wang, "Cyclone: A safe dialect of C," in *Proc. USENIX Annu. Tech. Conf.*, 2002, pp. 275–288.

PAUL C. VAN OORSCHOT, Associate Editor in Chief

EDITOR: Qun Li, liqun@cs.wm.edu

This article originally appeared in IEEE Internet Computing vol. 25, no. 3, 2021

The Emergence of Vehicle Computing

Sidi Lu and Weisong Shi, *Wayne State University, Detroit, MI, 48202, USA*

Connected and autonomous vehicles (CAVs) are poised to revolutionize the conventional transportation industry. In this article,we first introduce the vision of vehicle computing in the autonomous driving era and highlight that CAVs are the perfect computation platforms, so connected devices/things with limited computation capacities may rely on surrounding CAVs to perform complex computational tasks. Next,we depict several reasons why vehicle computing is essential and emerging, followed by four case studies, including in-vehicle delivery, in-vehicle meeting, in-vehicle entertainment, and in-vehicle augmented reality, to further illustrate vehicle computing. Finally, we conclude this article by listing several technical challenges related to vehicular communication, open APIs, computation hardware, energy consumption, computation offloading, as well as security and privacy.

CONNECTED AND AUTONOMOUS VEHICLES: FROM PRESENT TO FUTURE

The proliferation of communication and edge computing[1] has pushed the horizon of autonomous driving. Although technical obstacles, exorbitant costs, and social acceptability have still hindered large-scale production of connected and autonomous vehicles (CAVs), there has been an acceleration in the research and development (R&D) efforts to bring the idea of CAVs to fruition. For example, automakers spend more than 100 billion worldwide on R&D with around 5000 patents granted each year. Based on the recent considerable progress and disruptive technologies, optimists predict that by 2030, CAVs will be sufficiently reliable and commercially affordable to replace human driving.

This article envisions the next paradigm of future CAVs whose functionality will not only be limited to driving efficiently and safely in complex scenes. Instead, the future fully CAVs are expected to be universal computing platforms supporting daily life applications by providing efficient onboard computation for connected infrastructures. In this article, we introduce the concept of vehicle computing. We start from the analysis of why we need vehicle computing. Several case studies including in-vehicle delivery, in-vehicle meeting, in-vehicle entertainment, and in-vehicle augmented reality (AR) are introduced to further explain vehicle computing, followed by technical challenges waiting to address for the arrival of fully CAVs. We hope this article will gain attention from the automotive communities and inspire more research in vehicle computing.

1089-7801 © 2021 IEEE
Digital Object Identifier 10.1109/MIC.2021.3066076
Date of current version 18 June 2021.

VEHICLE COMPUTING

In this section, we give our definition and understanding of vehicle computing, and then we list several reasons why vehicle computing is important in the postautonomous driving era.

What is Vehicle Computing

Vehicle computing refers to the enabling technologies allowing computation to be performed on CAVs, which will serve as a computing platform for multiple CAV-related services. Different from vehicular networking,[2] which serves as the communication enabler for a myriad of applications related to vehicles and transportation, vehicle computing focuses on the computation functionality of CAVs and highlights that CAVs are the perfect computation platforms helping to analyze real-time data from in-vehicle sensors, and most importantly, from the surrounding connected devices/things, even when the vehicle is in the parking mode.

More specifically, the concept of vehicle computing is inspired by the fact that future CAVs will be equipped with powerful computing capability; therefore, connected devices/things with limited computation capacities can rely on nearby CAVs to perform complex computational tasks and deliver related results back to the end-users. For example, suppose a law enforcement officer equipped with a body-worn camera is on duty. The body-worn camera is collecting and sending video data to the surrounding law enforcement vehicle for latency-sensitive analytical applications, such as object detection. A warning will be sent by the vehicle when the officer is in a potentially dangerous situation. In this example, the law enforcement vehicle serves as the efficient computing platform based on the received data from the connected devices/things (i.e., body-worn camera) so that computing resources can be reasonably and effectively utilized, and the computation tasks can be completed on time.

Drawing from the definition of vehicle computing, we further introduce the future vehicle computing paradigm in Figure 1, which is driven by the communication of vehicle-to-infrastructure (V2I), vehicle-to-vehicle (V2V), and potentially vehicle-to-everything (V2X). V2X not only enables CAVs to communicate with the components of the traffic system (e.g., road-side units, cellular towers, traffic cameras, drones, scooters, and even cyclists or pedestrians), but also allows CAVs to communicate with external systems, i.e., elements of the surrounding environment (e.g., smart home sensors, industry IoT devices, health sensors, and edge servers).

Why Do We Need Vehicle Computing

Push to Clouds and Edge Servers

CAVs are equipped with enormous sensors, which could produce around one gigabyte of data per second and generate more than 11 TB of privacy-sensitive data on a daily basis. The quantity of data generated on CAVs is still growing, and the speed of data transportation is becoming the bottleneck when pushing data to clouds or edge servers for data analysis, which poses a significant challenge to provide latency-sensitive services. Besides, even if data are compressed in the CAVs before being sent out, the original sensitive data might be exposed, and it may create a potential threat of privacy leakage. Therefore, the bandwidth limitations, latency bottlenecks, and privacy concerns, in turn, calls for vehicle computing, a new computing paradigm to put the computing at the proximity of data. Previous work also demonstrated the potential benefits (such as the significant response time and energy reduction) by moving computing from the cloud to the data source.[3]

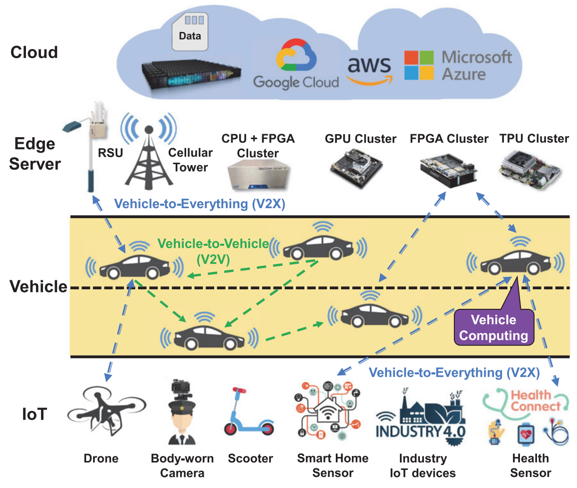


FIGURE 1. Vehicle computing paradigm.

Pull From IoT Devices

Nearly all types of electrical devices will become components of IoT and play the role of both data producers and consumers, such as body-worn cameras, scooters, and even Internet-connected bicycles. According to Cisco, the number of IoT worldwide devices will be around 500 billion by 2030. Such huge amounts of IoT devices will definitely produce enormous data, which hinders the execution of deep learning algorithms on the resource-constrained IoT devices. However, simply relying on traditional cloud computing cannot guarantee efficient data processing to handle all these generated data. In this context, we infer that IoT devices with limited computation capabilities will leverage the surrounding CAVs equipped with strong computing power to perform data processing on time, and we envision that vehicle computing will have big impact on automotive and IoT communities.

CASE STUDY

In this section, we introduce several promising case studies where vehicle computing could shine to further illustrate our vision of future CAVs.

In-Vehicle Delivery

We opine that the widespread of CAVs will be a key component of smart homes to assist people's daily life. For example, CAVs can provide a new, convenient, and secure in-vehicle delivery service when the customer are away from home. Today, Amazon, the world's largest online retailer, is taking the obvious next step by cooperating with mainstream automakers and launching early in-vehicle delivery services. Once the delivery driver reaches the vehicle parked in a publicly accessible place, the driver will send a request to remotely unlock the vehicle for delivery. After placing packages in cargo area or cabin, the driver will send a remote command to lock the vehicle again, and the customer will receive a final notification. Following this way, customers can receive packages safely even when they are not home.

In-Vehicle Meeting

In addition, since a fully CAV could achieve safe and reliable navigation by itself, there is no driver needed to focus on driving anymore. In this context, future CAVs are expected to provide other intelligent services such as providing efficient and smooth online meeting experiences in vehicles. Specifically, as the rapid development of wireless and sensor technologies enables secure and interoperable communications among vehicles, clouds, and devices/things (such as passengers' personal communication devices), we envision that the future CAVs are able to support in-vehicle meetings allowing people to share information and data without being physically present at home or office. Besides, people will be able to seamlessly attend the same meeting at home, in the vehicle, and in the office without being bothered by the repeated logout and login process, which really improves work efficiency and saves working time.

In-Vehicle Entertainment

Similarly, future CAVs have the potential to transform the way people travel by providing audio and video entertainment to enhance people's ride experience. MarketsandMarkets predicts that the in-vehicle entertainment market is estimated to reach USD 30.47 billion by 2022. Besides, starting in 2023, millions of Ford and Lincoln vehicles will be powered by Google's Android operating system to provide drivers with embedded Google applications and services. This evolution indicates that in-vehicle entertainment is on the rise. We envision in the era of fully autonomous driving that the passengers can select a variety of extended reality (XR) gaming via an interface and fully immerse themselves in the gaming experience. These XR games can provide real-time physical vehicle feedback, such as the driver's accelerating, stopping, and steering; therefore, each game experience is unique. Besides, thanks to V2V communication, passengers of different CAVs can play in-vehicle games together on the road, which will further increase the diversity of in-vehicle entertainment.

In-Vehicle AR

Moreover, we envision that AR technologies will be able to turn CAVs' windshields into movie screens, which will make the dreary journey be more interesting and secure by delivering passengers full-color graphics about their environment with a wide-viewing angle. Today, Civil Maps, a software provider for 3D maps, has revealed an AR experience for passengers, which can show passengers how a CAV equipped with AR displays navigates in the complex driving environment. Besides, Alibaba has invested $18 million in Way-Ray, a head-up display (HUD) company that released NAVION, the first holographic AR vehicle navigation system that can display travel details without wearing an AR helmet or glasses. When fully CAVs will come out, we opine that AR-enabled HUDs will be replaced by AR-enabled windshields, which can respond to voice commands and hand gestures.

TECHNICAL CHALLENGES

We have described four potential applications of vehicle computing in the previous section. To realize the vision of vehicle computing, we argue that the systems, algorithms, and network community need to work together. In this section, we will further summarize technical challenges in detail.

Vehicular Communication

It is estimated that by 2025, there will be 470 million CAVs on highways worldwide, generating 280 petabytes of data. Besides, when the CAV is driving in the urban area at a speed of 40 kilometers per hour, the execution time of each real-time task should be less than 100 milliseconds. However, performing efficient computation based on such a big amount of data is challenging as it requires ultrareliable and low-latency communications (URLLC) to accommodate multiple services.

The recent proliferation in communication mechanisms, such as dedicated short range communication (DSRC), long-term evolution (LTE), cellular-vehicle-to-everything (C-V2X), and WiFi, has enabled CAVs to obtain information from other vehicles, clouds, and connected devices/things.[4] Particularly, with decades of development history, DSRC has been widely deployed, but it has issues like small coverage and low throughput. In contrast, WiFi and LTE provide more bandwidth but perform poorly in the mobile environment. With the recently developed access technology, C-V2X could tackle communication issues due to high mobility and vehicular density scenarios. However, C-V2X is not affordable and widely deployed compared with DSRC. Therefore, the development of communication mechanisms still has a long way to go.

Open APIs

Machine learning-based applications are vastly utilized by CAVs. Unfortunately, there are very limited public computing platforms that support vehicular data analytic and processing. Except for Baidu's Apollo, many companies, such as Ford and General Motors, are working on their proprietary platforms. Moreover, although Apollo is open-source, it is neither scalable nor suitable for future CAVs with plenty of third-party services.

In contrast to the proprietary platform, the open-source platforms that offer free APIs and real-field vehicle data to the researchers and developers are needed, as the open APIs allow communities to deploy and evaluate applications in the real environment. Recently, BlackBerry and AWS are joining forces to develop BlackBerry IVY, a scalable, cloud-connected software platform that will allow automakers to improve operations of CAVs with new BlackBerry QNX and AWS technology. Besides, researchers proposed Open Vehicular Data Analytics Platform (OpenVDAP),[5] which is a full-stack hardware/software platform providing a public edge-aware application library. More open APIs are needed for CAVs to push the development of the third-party services.

Computation Hardware

Nowadays, representative automotive-grade computation hardware of CAVs is being designed based on graphic processor unit (GPU), field programmable gate arrays (FPGA), digital signal processor (DSP), and application-specific integrated circuit (ASIC) with improved processing speed and energy efficiency, such as NVIDIA DRIVE AGX and Texas Instruments' TDA.

However, to design a hardware system for vehicle computing scenarios, there are several open problems waiting to be addressed. First, it is important to figure out the maximum speed that the hardware can achieve with limited processing power. Second, how to efficiently manage heterogeneous computation resources and dynamically schedule applications has deserved researchers' attention. Besides, it is also essential to evaluate how suitable a hardware system is for a specific application scenario. Moreover, a level 4 CAV may cost up to 300,000 dollars, in which the sensors and computing platform cost almost two-thirds of the total price. Therefore, it is also necessary to design a reasonably priced hardware system.

Energy Consumption

With enormous sensors and complex algorithms implemented on CAVs, energy consumption has become a big problem for CAVs. Take the NVIDIA Drive PX Pegasus as an example, it consumes 320 INT8 TOPS of AI computing power with a budget of 500 watts. Moreover, if a replicated system is installed to ensure the reliability of autonomous driving, the total power consumption may be as high as nearly 2000 W.

Besides, take the electric vehicle (EV) as an example, suppose that in the United States, the total mileage of each EV is composed of 55% of city mileage and 45% of highway mileage, and each EV travels on cities and highways at a speed of 31 mph and 56 mph, respectively. In this case, the annual energy consumption of EVs nationwide for computation is around 180 terawatt-hours.[6] It is reported that Google data centers now use around 12 terawatt-hours of electricity per year,[7] so we infer that the national energy consumption of EVs is approximately equal to the total energy consumption of 15 representative technology companies' data centers each year.

Therefore, how to deal with a large amount of energy consumption is an important issue. Moreover, since most of the energy is consumed by the electric motor of the vehicle, it is necessary to jointly design the battery, energy management system, and computing system to realize energy-efficient autonomous driving.

Computation Offloading

Although future CAVs will be endowed with server-level computing power to process sensing data, it becomes evident that safe and reliable autonomous driving requires effective V2X computations to transmit critical information. Accordingly, vehicles and connected devices/things usually work together to process the sensing data, extend their sensing capabilities, and coordinate their decisions.

Nonetheless, collaborative computing between CAVs and connected devices/things is not always feasible due to the latency and reliability constraints. Considering the heterogeneity of the computing capabilities and the interdependency of computing tasks, researchers have formulated optimization problems for task scheduling. Lots of works has focused on task offloading algorithms to optimize the computation offloading. For example, a resource allocation method is proposed to optimize the performance of task offloading when the computation requirement is unknown.[8] Similarly, Tran *et al.*[9] propose a task offloading model to optimize the cost of the computation. However, all the work is based on simulations, and the evaluation in the real-world application scenarios is still missing.

Security and Privacy

The security of CAVs has evolved from the hardware damage of conventional vehicles to comprehensive security with multidomain knowledge.[10] Here, we introduce several security problems strongly related to CAVs including the mainstream attacking methods.

Sensing Security

The security of sensors is of paramount importance. Generally, jamming attacks and spoofing attacks are two main attacks for various sensors. For instance, a spoofing attack generates interference signals, which can cause the vehicle to capture fake obstacles. Hence, effective protection mechanisms for sensing security are desired.

Data Security

Data security denotes preventing data leakage from the perspectives of transmission and storage. How to protect real-time and historical data is waiting for more advanced solutions.

Communication Security

Communication security includes the security of internal communication (such as CAN, LIN, and FlexRay) and external communication that has been studied in VANETs with V2X communications. Although cryptography is a frequently used solution, the usage of cryptography is limited due to the high computational cost.

Control Security

With vehicles' electronification, drivers could control their vehicles (e.g., open the door) through apps or voice. However, this also leads to new attack surfaces with various attack methods, including jamming attacks, replay attacks, etc.

Privacy

CAVs rely heavily on data from the surrounding environment and generate personalized driving data, which usually contains private information. For example, an attacker can obtain the location information directly from the captured GPS data. Therefore, more data desensitization methods are needed to protect the privacy of drivers and passengers.

CONCLUSION

In this article, we first present the vision for vehicle computing in the connected and autonomous driving era. Then, we depict several reasons why vehicle computing is important and emerging, followed by several case studies to further illustrate our vision. Finally, we conclude the article by listing several technical challenges.

REFERENCES

1. W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.
2. G. Karagiannis *et al.*, "Vehicular networking: A survey and tutorial on requirements, architectures, challenges, standards and solutions," *IEEE Commun. Surv. Tut.*, vol. 13, no. 4, pp. 584–616, Oct–Dec. 2011.
3. S. Yi, Z. Hao, Z. Qin, and Q. Li, "Fog computing: Platform and applications," in *Proc. 3rd IEEE Workshop Hot Top. Web Syst. Technol*, 2015, pp. 73–78.
4. K. Z. Ghafoor, M. Guizani, L. Kong, H. S. Maghdid, and K. F. Jasim, "Enabling efficient coexistence of DSRC and C-V2X in vehicular networks," *IEEE Wirel. Commun.*, vol. 27, no. 2, pp. 134–140, Apr. 2020.
5. Q. Zhang *et al.*, "OpenVDAP: An open vehicular data analytics platform for CAVs," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst.*, 2018, pp. 1310–1320.
6. Teraki, "Autonomous cars' big problem: The energy consumption of edge processing reduces a car's mileage with up to 30%," May 2019, [Online]. Available: https://medium.com/@teraki/energy-consumption-required-by-edge-computing-reduces-a-autonomous-cars-mileage-with-up-to-30-46b6764ea1b7
7. R. Bryce, "How Google powers its 'monopoly' with enough electricity for entire countries," Oct. 2020, [Online]. Available: https://www.forbes.com/sites/robertbryce/2020/10/21/googles-dominance-is-fueled-by-zambia-size-amounts-of-electricity/?sh=19fc3bd168c9
8. N. Eshraghi and B. Liang, "Joint offloading decision and resource allocation with uncertain task computing requirement," in *Proc. IEEE Conf. Comput. Commun*, 2019, pp. 1414–1422.
9. T. X. Tran, K. Chan, and D. Pompili, "COSTA: Cost-aware service caching and task offloading assignment in mobile-edge computing," in *Proc. 16th Annu. IEEE Int. Conf. Sens., Commun., Netw.*, 2019, pp. 1–9.
10. L. Liu *et al.*, "Computing systems for autonomous driving: State-of-the-art and challenges," *IEEE Internet Things J.*, to be published, doi: 10.1109/JIOT.2020.3043716.

SIDI LU is with the Department of Computer Science, Wayne State University, Detroit, MI, USA. She is the corresponding author of this article. Contact her at: lu.sidi@wayne.edu.

WEISONG SHI is with the Department of Computer Science, Wayne State University, Detroit, MI, USA. Contact him at: weisong@wayne.edu.

IEEE COMPUTER SOCIETY D&I FUND

Drive Diversity & Inclusion in Computing

Supporting projects and programs that positively impact diversity, equity, and inclusion throughout the computing community.

Do you have a great idea for new programs that will positively impact diversity, equity, and inclusion throughout the computing community?

The IEEE Computer Society Diversity & Inclusion Committee seeks proposals for projects, programs, and events that further its mission. New programs that deliver education, outreach, and support, including, but not limited to, mentoring programs at conferences, panel discussions, and webinars, are welcomed.

Help propel the Computer Society's D&I programs—submit a proposal today!

https://bit.ly/CS-Diversity-CFP

Donations to the IEEE Computer Society D&I Fund are welcome!

IEEE COMPUTER SOCIETY

IEEE Foundation

EDITOR: Ron Vetter, University of North Carolina Wilmington; vetterr@uncw.edu

DEPARTMENT: SPOTLIGHT ON TRANSACTIONS

This article originally appeared in Computer vol. 54, no. 2, 2021

Secure V2V and V2I Technologies for the Next-Generation Intelligent Transportation Systems

Sudip Mittal, *University of North Carolina Wilmington*

This installment of Computer*'s series highlighting the work published in IEEE Computer Society journals comes from* IEEE Transactions on Services Computing.

Vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) technologies are transforming the digital landscape. The automotive industry is shifting into a new digital age, where connected vehicles and smart cars are starting to collaborate among themselves with less reliance on human drivers. This communication is especially useful when considering their benefits toward smart cities. Smart vehicles can exchange information with each other, physical infrastructure like roadside units, and even potentially pedestrians. These use cases present bountiful opportunities for cities to address a number of issues, from traffic management to even the prevention of potential collisions. Despite the benefits, V2V and V2I communication technologies also present a broad attack surface for cybercriminals. Some examples include stealing private data, remotely hijacking a vehicle, and coordinating roadside infrastructure attacks.

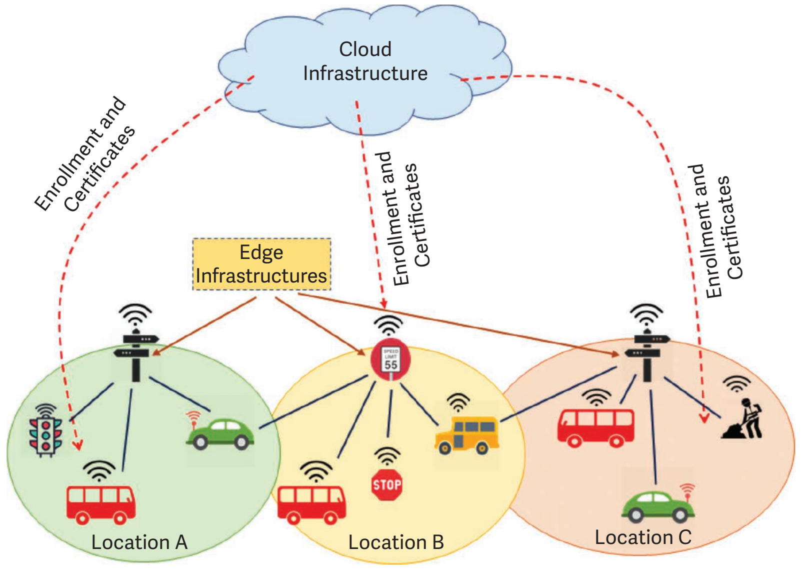

Cloud Infrastructure
Enrollment and Certificates
Enrollment and Certificates
Enrollment and Certificates
Edge Infrastructures
55
STOP
Location A
Location B
Location C

FIGURE 1. The proposed trusted cloudlet architecture. (Taken from Gupta et al.[1])

M. Gupta et al.[1] present an approach to securing V2V and V2I communication by utilizing cloudlets to ensure the confidentiality, integrity, and authentication of messages across a system. In addition, they discuss an attribute-based access control model for V2V and V2I called the *attribute-based intelligent transportation system* (*AB-ITS*). The proposed cloudlet architecture is depicted in Figure 1. Trusted edge infrastructures produced by city administrators will operate as intermediaries between vehicles and entities inside the city's geographic range by relaying secured messages. At the edge, messages are validated by a set of predetermined security policies before being forwarded across the interconnected network. Figure 2 illustrates a conceptual AB-ITS model. The attributes developed in the AB-ITS are supported by the cloudlets. A source

Digital Object Identifier 10.1109/MC.2020.3042227
Date of current version: 11 February 2021

initiates operations on cloudlets and can be a set of vehicles, a transportation infrastructure, or administrative users. Trusted cloudlets (TCs) enroll devices into a system through the use of a traditional public key infrastructure scheme. Target vehicles (VT) and source vehicles must be a part of the same TC to communicate. Authorization policies and attributes define operations for the overall secure functioning of the ecosystem. Policies and attributes are also dynamic in nature and can shift to fit changing circumstances or communication preferences in a city.

A proof-of-concept implementation of the AB-ITS was simulated on the Amazon Web Services Internet of Things platform. The authors modeled situations such as ice-on-road and compromised rogue vehicles. The performance of the model can be measured by the execution time of attribute-based security policies against the number of vehicles associated with a cloudlet. The authors found that the total trip time was comparable to that for a peer-to-peer ITS despite variations due to network traffic and latency. In a large city, more cloudlets and infrastructure devices can be installed to reduce the crowding of vehicles within one cloudlet, improving the execution time.

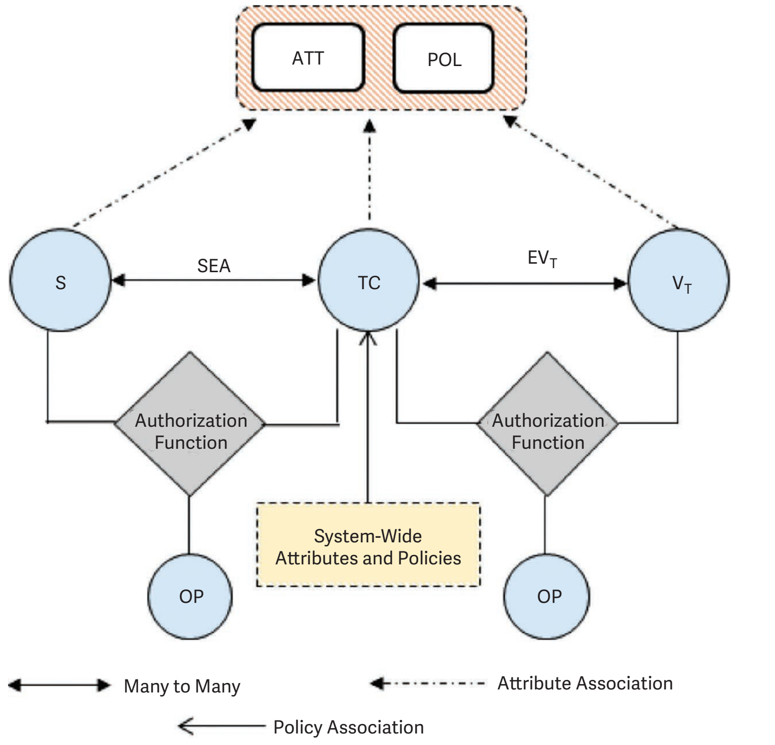


FIGURE 2. An AB-ITS communication model. ATT: attributes; POL: policies; S: source; TC: trusted cloudlets; VT: target vehicles; OP: operations; SEA: source entity attribute relations; EVT: vehicle to trusted cloudlet relations. (Taken from Gupta et al.[1])

THE AUTOMOTIVE INDUSTRY IS SHIFTING INTO A NEW DIGITAL AGE, WHERE CONNECTED VEHICLES AND SMART CARS ARE STARTING TO COLLABORATE AMONG THEMSELVES WITH LESS RELIANCE ON HUMAN DRIVERS.

REFERENCE

1. M. Gupta, J. Benson, F. Patwa, and R. Sandhu, "Secure V2V and V2I communication in intelligent transportation using cloudlets." *IEEE Trans. Services Comput.*, vol. 13, no. 14, pp. 1–13, Sept. 22, 2020. doi: 10.1109/TSC.2020.3025993.

SUDIP MITTAL is an assistant professor in the Department of Computer Science at the University of North Carolina Wilmington, Wilmington, North Carolina, 28403, USA. Contact him at mittals@uncw.edu.

EDITORS: Beatriz Sousa Santos, bss@ua.pt
Ginger Alford, alfordg@smu.edu

DEPARTMENT: EDUCATION

This article originally appeared in IEEE Computer Graphics and Applications vol. 40, no. 2, 2020

Teaching Clustering Algorithms With EduClust: Experience Report and Future Directions

Johannes Fuchs, *University of Konstanz*

Petra Isenberg and Anastasia Bezerianos, *Université Paris-Saclay, CNRS, Inria, LRI*

Matthias Miller and Daniel A. Keim, *University of Konstanz*

We share our experiences teaching university students about clustering algorithms using EduClust, *an online visualization we developed.* EduClust *supports professors in preparing teaching material and students in visually and interactively exploring cluster steps and the effects of changing clustering parameters. We used* EduClust *for two years in our computer science lectures on clustering algorithms and share our experience integrating the online application in a data science curriculum. We also point to opportunities for future development.*

We are currently seeing an immense increase in online learning platforms and sharing of teaching material.[1] We implemented *EduClust* (see Figure 2) to reduce the considerable effort in creating high-quality teaching material and to encourage learning in and outside the classroom. *EduClust* is an easily accessible online visualization application, which supports dynamic teaching and learning of clustering algorithms.[2] Simple two-dimensional data representations like scatterplots are used to show clustering behavior. We added animations to communicate changes between algorithmic steps. Different algorithms can be applied to various datasets and can be steered by changing input parameters or distance metrics. Additionally, further details about the algorithms are provided in a separate panel showing pseudocode, algorithmic complexity, and hyperparameters.

For two years, we used *EduClust* in our teaching routine. Based on our experiences with the software, we provide the interested reader with some guidance on preparing and organizing teaching material (e.g., slides and assignments) together with ideas about how to include the software in classroom settings (e.g., hands-on sessions with students). Given the positive feedback from our students using the software, we want to motivate similar development and research in this area.

Digital Object Identifier 10.1109/MCG.2020.2970560
Date of current version 28 February 2020.

TEACHING SCENARIOS WITH *EduClust*

EduClust is accessible online (educlust.dbvis.de/) and comes with nine different clustering implementations and an initial pool of datasets. Teachers and students can start right away using the software. In our data mining lecture at the University of Konstanz, we teach several different clustering algorithms. Our learning goals are based on Krathwohl's revised version of Bloom's educational objectives.[3] They comprise simple cognitive processes like *remembering* for which clustering algorithms exist and *understanding* the different categories the algorithms fall into, as well as the single steps of the algorithmic behavior. We want students to be able to

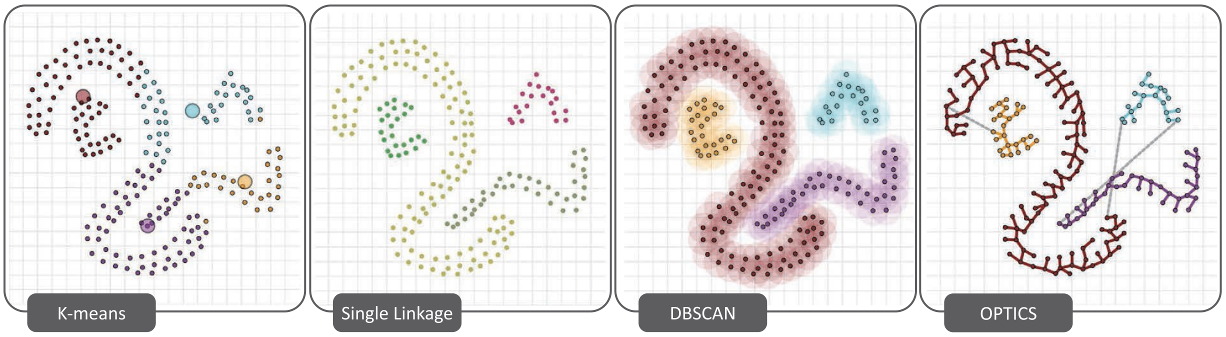
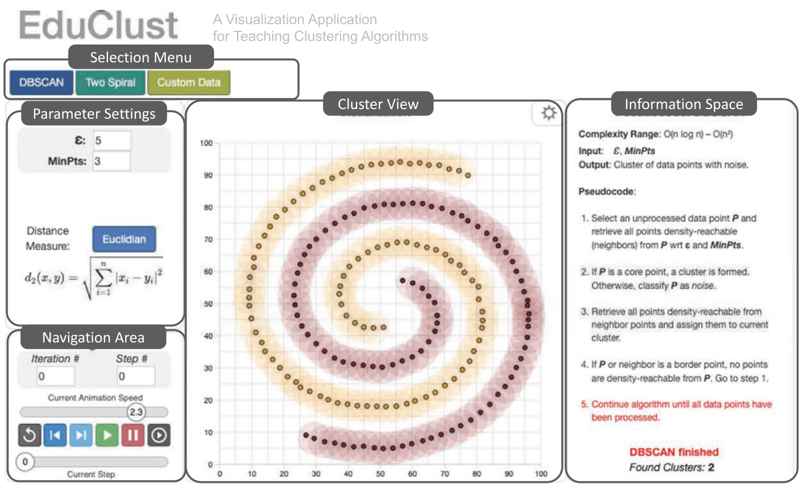


FIGURE 1. Application overview: Algorithms and datasets can be selected in the Selection Menu (top), parameters and animations can be adjusted in the Navigation Area and Parameter Settings (left), detailed text descriptions are displayed in the Information Space (right), and the clustering behavior is visualized in the Cluster View (center).

apply the clustering algorithms to data in a meaningful way and *analyze* the influence of input parameters or distance measures on the clustering result. Ultimately we want students to *evaluate* (see Figure 1) the performance of clustering algorithms in certain situations and *create* scenarios, in which algorithms fail or outperform others. With *EduClust*, we were able to teach even complex cognitive processes and reduce the preparation time of the lecturer to a minimum.



ε: 5
MinPts: 3

Distance Measure: Euclidian

$d_2(x, y) = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2}$

Iteration # 0
Step # 0

Complexity Range: O(n log n) – O(n²)
Input: *Ɛ*, ***MinPts***
Output: Cluster of data points with noise.

Pseudocode:

1. Select an unprocessed data point ***P*** and retrieve all points density-reachable (neighbors) from ***P*** wrt ε and ***MinPts***.
2. If ***P*** is a core point, a cluster is formed. Otherwise, classify ***P*** as *noise*.
3. Retrieve all points density-reachable from neighbor points and assign them to current cluster.
4. If ***P*** or neighbor is a border point, no points are density-reachable from ***P***. Go to step 1.
5. Continue algorithm until all data points have been processed.

DBSCAN finished
Found Clusters: **2**

FIGURE 2. Four of the nine clustering algorithms and their visualization supported in EduClust: (a) k-means clustering centers shown as circles. Shapes in this artificial dataset are not separated well. (b) In Single Linkage, a dendrogram from the hierarchical clustering (not shown here) was used to determine the effective horizontal cut to differentiate each shape. (c) DBSCAN algorithm uses an ϵ-distance, which is represented using blurry circles. (d) Visualization shows the spanning tree of the OPTICS algorithm.

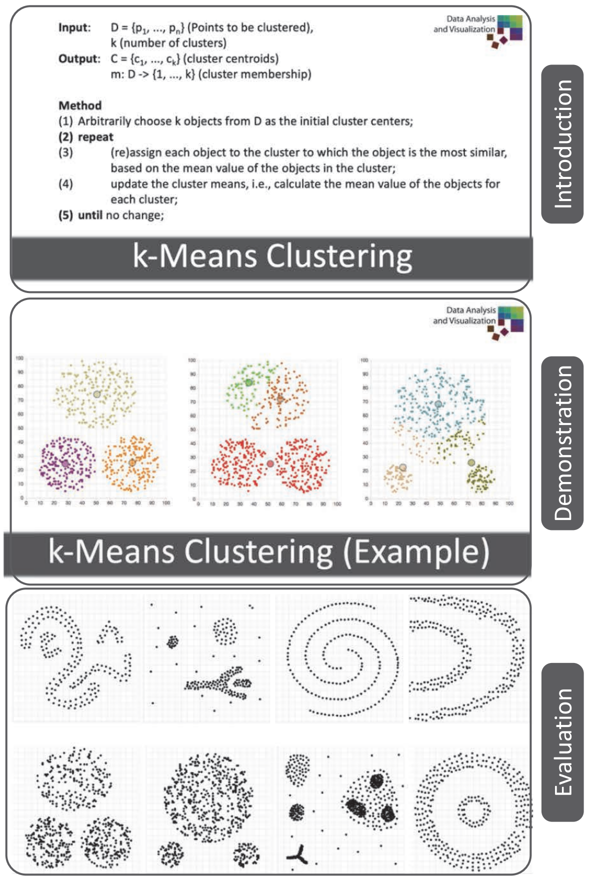
Preparing Slides

The preparation of slides to show clustering steps can be a tedious task. To visually show changes over time, multiple intermediate steps of the algorithmic behavior need to be drawn out and displayed, ideally with animation.

With *EduClust*, one can use an export mechanism for individual clustering animations and the details about the algorithm provided in the information space. Lecturers just have to decide which algorithms, hyperparameters, and datasets they want to include in the slide deck. They run the software once and export the displayed animations in graphics interchange format to use in their slides. For details about the algorithmic behavior, complexity, or input parameters, *EduClust* provides ample text that can be copied on slides as well.

During the Lecture

To profit from *EduClust* during the lecture, we found it useful to split the session into three parts. First, a theoretical introduction to a new algorithm; next, a hands-on session; and finally a group discussion of advantages and disadvantages of different algorithms.


Data Analysis and Visualization
Introduction
Data Analysis and Visualization
Demonstration
k-Means Clustering (Example)
Evaluation

```
Input:   D = {p_1, ..., p_n} (Points to be clustered),
         k (number of clusters)
Output:  C = {c_1, ..., c_k} (cluster centroids)
         m: D -> {1, ..., k} (cluster membership)

Method
(1) Arbitrarily choose k objects from D as the initial cluster centers;
(2) repeat
(3)      (re)assign each object to the cluster to which the object is the most similar,
         based on the mean value of the objects in the cluster;
(4)      update the cluster means, i.e., calculate the mean value of the objects for
         each cluster;
(5) until no change;
```

k-Means Clustering

FIGURE 3. Our instructional material consists of three parts: first, introduction slides with pseudocode; second, a live demonstration of the clustering behavior (animations exported with EduClust); third, an evaluation of the clustering results using datasets with different characteristics (in-class exploration with EduClust).

In our lectures, we always introduce new clustering algorithms showing slides with text information and pseudocode first, followed by a moderated animation generated by *EduClust* (see Figure 3). Students can see the algorithm in action, understand the individual clustering steps, and relate to the previously shown text descriptions. This first introduction is meant to support the cognitive processes *remember* and *understand*.

In the second part of the lecture, students use *EduClust* on their own to *apply* the algorithms to different datasets and *analyze* their peculiarity. Students, thus, experience the influence of changing input parameters and cluster characteristics. The duration of these individual hands-on sessions depends on the complexity of the algorithm.

In the third part, we put clustering algorithms into context with each other (see Figure 1). The lecturer starts a discussion by bringing up a dataset with specific characteristics. Students then discuss whether or not clustering algorithms are capable of separating data points into clusters. The lecturer and students use *EduClust* to try algorithms with various input parameters and discuss their advantages and disadvantages. Thereby, students *evaluate* the usefulness of algorithms and understand their individual application areas.

We found that this lecture structure covers nearly all cognitive processes to support student learning. However, we recommend to accompany the session with an assignment sheet to also support *create* as the another cognitive process.

Preparing an Assignment

Our assignments are designed to generate a deep engagement with specific clustering processes. We ask questions that require students to *apply* algorithms, *analyze* the consequences when changing input parameters, or *evaluate* different clustering techniques given a certain dataset. To further increase the learning rate, we also include questions, in which students have to *create* datasets being suitable for the one algorithm but not for the others. In such scenarios, students have to understand details of the algorithms to come to a solution. Trial and error usually fails due to the complexity of the problem space with many different variables, e. g., input parameters, clustering algorithms, or distance metrics.

Student Assessment

EduClust supports the export and import of data files in the json format. This feature can facilitate the correction of submissions. When students have to *create* datasets for their assignment, they can export them and email their result to the lecturer. The lecturer can use *EduClust* to import the dataset and check for correctness.

Summary of Benefits

Although not exhaustive, *EduClust* covers the most prominent clustering algorithms and provides a visual categorization based on their clustering behavior. Lecturers can do live demonstrations of the clustering behavior of individual algorithms and use *EduClust* to prepare teaching material. *EduClust* offers datasets covering various cluster characteristics, which can be used together with all implemented clustering algorithms. The influence and importance of choosing input parameters wisely can be shown by running the same algorithm on the same dataset with varying input parameters. During the lecture, multiple clustering algorithms can be compared using the same dataset, revealing the benefits of each clustering algorithm. Both the description of algorithmic steps (in text), and a sequence of images showing these steps on a dataset, can be exported and added to traditional teaching material.

Students can apply clustering algorithms without implementation effort to various datasets and rerun the same algorithm multiple times using different input parameters. While running the algorithm, the underlying pseudocode is displayed in the information space. The selection of different algorithms and datasets help students to evaluate the performance of the respective algorithms. Finally, students can create individual datasets to be clustered with all implemented algorithms.

FUTURE RESEARCH DIRECTIONS

Qualitative evaluations showed that students are willing to use *EduClust* in their learning routine.[2] Currently, *EduClust* is limited to nine clustering algorithms, but we will extend it to include cluster quality measures and additional algorithms like DENCLUE.

Given the positive feedback from our students, we also see a lot of potential for applying what we learned to different categories of algorithms. A promising starting point could be decision trees. In addition, we would like to use *EduClust* as a motivation to establish a new research direction called teachable AI. While explainable AI gets a lot of research attention, respective applications focus on understanding the algorithmic behavior of individual architectures. We would like to argue for further research toward experiencing the entire inner workings of multiple algorithms together with the consequences of changing parameters and the possibility to evaluate different approaches on the same dataset, along the lines of explAIner[4] but with a focus on teaching ML algorithms from a professor and a student perspective.

REFERENCES

1. I. E. Allen and J. Seaman, *Changing Course: Ten Years of Tracking Online Education in the United States.* Babson Park, MA, USA: Babson Survey Research Group, 2013.
2. J. Fuchs, P. Isenberg, A. Bezerianos, M. Miller, and D. Keim, "Educlust-a visualization application for teaching clustering algorithms," in *Proc. Eurographics—Educ. Papers*, 2019. [Online]. Available: https://dx.doi.org/10.2312/eged.20191023
3. D. R. Krathwohl, "A revision of bloom's taxonomy: An overview," *Theory Into Practice*, vol. 41, no. 4, pp. 212–218, 2002.
4. T. Spinner, U. Schlegel, H. Schäfer, and M. El-Assady, "Explainer: A visual analytics framework for interactive and explainable machine learning," *IEEE Trans. Vis. Comput. Graphics*, vol. 26, no. 1, pp. 1064–1074, Jan. 2020.

JOHANNES FUCHS is a Research Scientist and Lecturer with the University of Konstanz, Konstanz, Germany. Contact him at fuchs@dbvis.inf.uni-konstanz.de.

PETRA ISENBERG is a Research Scientist with Inria, Rocquencourt, France in the Aviz team. Contact her at petra.isenberg@inria.fr.

ANASTASIA BEZERIANOS is an Associate Professor with University Paris-Saclay, Essonne, France, and part of the Inria ILDA team. Contact her at anastasia.bezerianos@lri.fr.

MATTHIAS MILLER is a Research Associate and is currently working toward the Ph.D. degree with the University of Konstanz, Konstanz, Germany. Contact him at miller@dbvis.inf.uni-konstanz.de.

DANIEL A. KEIM is a Full Professor and the Head of the Information Visualization and Data Analysis Research Group, University of Konstanz, Konstanz, Germany. Contact him at keim@uni-konstanz.de.

Contact department editors Beatriz Sousa Santos at bss@ua.pt and Ginger Alford at alfordg@smu.edu.

IEEE Computer Society Has You Covered!

WORLD-CLASS CONFERENCES — Stay ahead of the curve by attending one of our 210 globally recognized conferences.

DIGITAL LIBRARY — Easily access over 800k articles covering world-class peer-reviewed content in the IEEE Computer Society Digital Library.

CALLS FOR PAPERS — Discover opportunities to write and present your ground-breaking accomplishments.

EDUCATION — Strengthen your resume with the IEEE Computer Society Course Catalog and its range of offerings.

ADVANCE YOUR CAREER — Search the new positions posted in the IEEE Computer Society Jobs Board.

NETWORK — Make connections that count by participating in local Region, Section, and Chapter activities.

Explore all of the member benefits at www.computer.org today!

IEEE COMPUTER SOCIETY

IEEE

Conference Calendar

IEEE Computer Society conferences are valuable forums for learning on broad and dynamically shifting topics from within the computing profession. With over 200 conferences featuring leading experts and thought leaders, we have an event that is right for you. Questions? Contact conferences@computer.org.

FEBRUARY

7 February

- FOCS (IEEE Symposium on Foundations of Computer Science), Denver, USA

12 February

- CGO (Int'l Symposium on Code Generation and Optimization), Seoul, South Korea
- HPCA (IEEE Int'l Symposium on High-Performance Computer Architecture), Seoul, South Korea

26 February

- VLSID (Int'l Conf. on VLSI Design and Int'l Conf. on Embedded Systems), virtual

MARCH

12 March

- ICSA (IEEE Int'l Conf. on Software Architecture), Honolulu, USA
- VR (IEEE Conf. on Virtual Reality and 3D User Interfaces), Christchurch, New Zealand

15 March

- CSASE (Int'l Conf. on Computer Science and Software Eng.), Duhok, Iraq
- SANER (IEEE Int'l Conf. on Software Analysis, Evolution, and Reengineering), Honolulu, USA

21 March

- PerCom (IEEE Int'l Conf. on Pervasive Computing and Communications), Pisa, Italy

30 March

- WONS (Wireless On-Demand Network Systems and Services Conf.), Oppdal, Norway

APRIL

4 April

- ICST (IEEE Int'l Conf. on Software Testing, Verification and Validation), virtual

11 April

- PacificVis (IEEE Pacific Visualization Symposium), Tsukuba, Japan

25 April

- VTS (IEEE VLSI Test Symposium), San Diego, USA

MAY

4 May

- ICCPS (ACM/IEEE Int'l Conf. on Cyber-Physical Systems), Milano, Italy
- RTAS (IEEE Real-Time and Embedded Technology and Applications Symposium), Milano, Italy

9 May

- ICDE (IEEE Int'l Conf. on Data Eng.), virtual

15 May

- FCCM (IEEE Int'l Symposium on Field-Programmable Custom Computing Machines), New York, USA

16 May

- ICFEC (IEEE Int'l Conf. on Fog and Edge Computing), Messina, Italy

17 May

- ISORC (Int'l Symposium On Real-Time Distributed Computing), Västerås, Sweden

18 May

- ISCV (Int'l Conf. on Intelligent Systems and Computer Vision), Fez, Morocco
- ISMVL (IEEE Int'l Symposium on Multiple-Valued Logic), Dallas, USA

19 May

- SELSE (IEEE Workshop on Silicon Errors in Logic – System Effects), virtual

21 May

- ICSE (IEEE/ACM Int'l Conf. on Software Eng.), Pittsburgh, USA

22 May

- SP (IEEE Symposium on Security and Privacy), San Francisco, USA

23 May

- ETS (IEEE European Test Symposium), Barcelona, Spain

25 May

- SERA (IEEE/ACIS Int'l Conf. on Software Eng., Management and Applications), Las Vegas, USA

30 May
- DCOSS (Int'l Conf. on Distributed Computing in Sensor Systems), Los Angeles, USA
- IPDPS (IEEE Int'l Parallel & Distributed Processing Symposium), Lyon, France

JUNE

6 June
- EuroS&P (IEEE European Symposium on Security and Privacy), Genoa, Italy
- MDM (IEEE Int'l Conf. on Mobile Data Management), Paphos, Cyprus

11 June
- ISCA (ACM/IEEE Int'l Symposium on Computer Architecture), New York, USA

14 June
- WoWMoM (IEEE Int'l Symposium on a World of Wireless, Mobile and Multimedia Networks), Belfast, UK

19 June
- CVPR (IEEE/CVF Conf. on Computer Vision and Pattern Recognition), New Orleans, USA

25 June
- CSCLOUD (IEEE Int'l Conf. on Cyber Security and Cloud Computing), Xi'an, China

26 June
- ICIS (IEEE/ACIS Int'l Conf. on Computer and Information Science), Zhuhai, China

27 June
- COMPSAC (IEEE Computers, Software, and Applications Conf.), Torino, Italy
- DSN (IEEE/IFIP Int'l Conf. on Dependable Systems and Networks), Baltimore, USA
- HOST (IEEE Int'l Symposium on Hardware Oriented Security and Trust), McLean, Virginia, USA

JULY

1 July
- ICALT (IEEE Int'l Conf. on Advanced Learning Technologies), Bucharest, Romania

6 July
- ISVLSI (IEEE Computer Society Symposium on VLSI), Nicosia, Cyprus

10 July
- ICDCS (IEEE Int'l Conf. on Distributed Computing Systems), Bologna, Italy

11 July
- ICME (IEEE Int'l Conf. on Multimedia and Expo), Taipei, Taiwan

21 July
- CBMS (IEEE Int'l Symposium on Computer-Based Medical Systems), Shenzhen, China

AUGUST

1 August
- ICCP (Int'l Conf. on Computational Photography), Pasadena, USA

2 August
- MIPR (IEEE Int'l Conf. on Multimedia Information Processing and Retrieval), virtual

4 August
- BCD (IEEE/ACIS Int'l Conf. on Big Data, Cloud Computing, and Data Science Eng.), Danang, Vietnam

7 August
- CSF (IEEE Computer Security Foundations Symposium), Haifa, Israel

9 August
- IRI (IEEE Int'l Conf. on Information Reuse and Integration for Data Science), virtual

15 August
- RE (IEEE Int'l Requirements Eng. Conf.), Melbourne, Australia

SEPTEMBER

6 September
- CLUSTER (IEEE Int'l Conf. on Cluster Computing), Heidelberg, Germany

12 September
- ARITH (IEEE Symposium on Computer Arithmetic), virtual

IEEE COMPUTER SOCIETY JOBS BOARD

Evolving Career Opportunities Need Your Skills

Explore new options—upload your resume today

www.computer.org/jobs

Changes in the marketplace shift demands for vital skills and talent. The IEEE Computer Society Jobs Board is a valuable resource tool to keep job seekers up to date on the dynamic career opportunities offered by employers.

Take advantage of these special resources for job seekers:

- JOB ALERTS
- TEMPLATES
- WEBINARS
- CAREER ADVICE
- RESUMES VIEWED BY TOP EMPLOYERS

No matter what your career level, the IEEE Computer Society Jobs Board keeps you connected to workplace trends and exciting career prospects.

IEEE COMPUTER SOCIETY

IEEE