Trends in Software and Control

Sanz, Ricardo; Årzén, Karl-Erik

Published in:
Control Systems Magazine

DOI:
10.1109/MCS.2003.1200238

2003

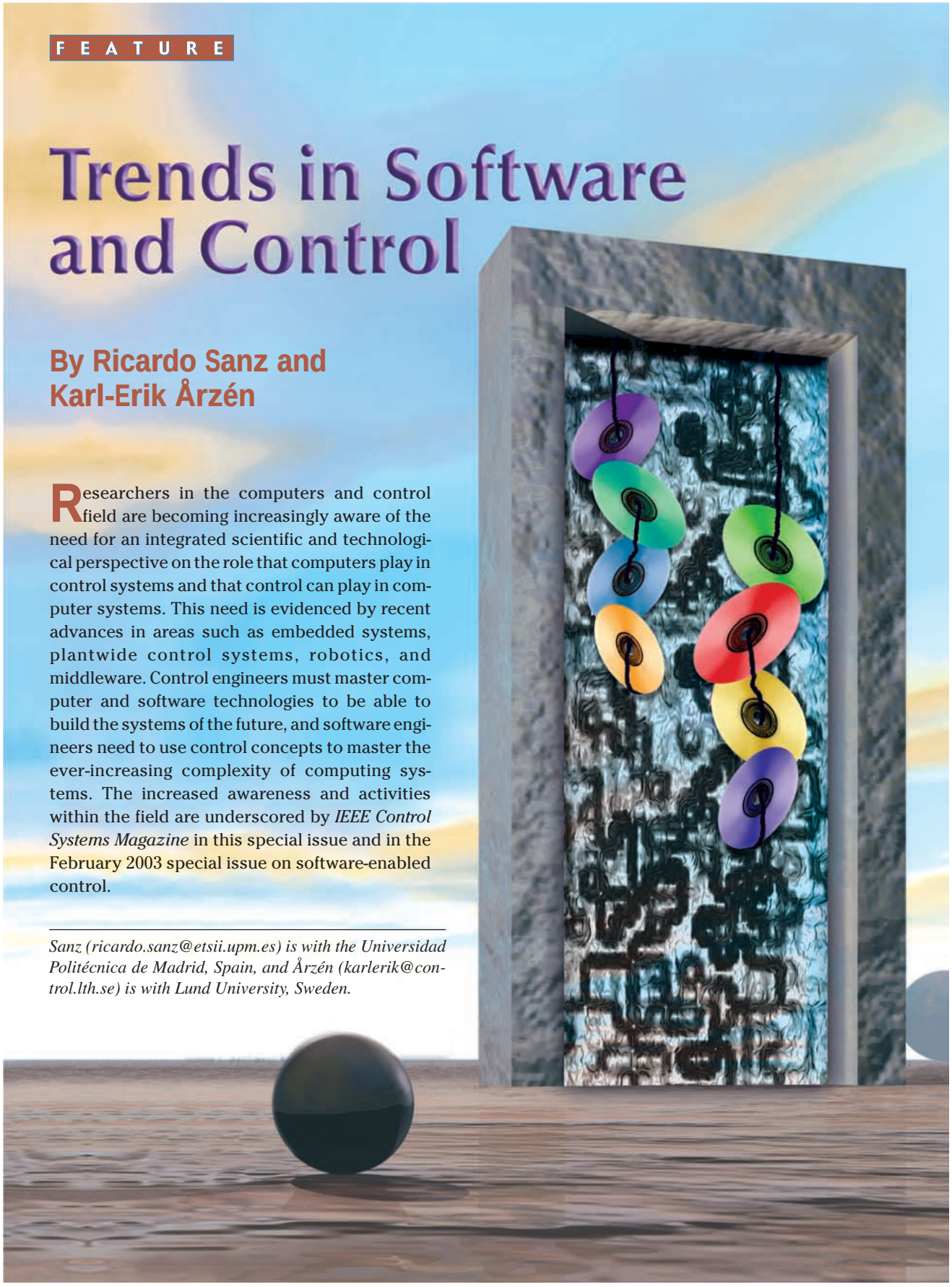
FEATURE

Trends in Software and Control

By Ricardo Sanz and Karl-Erik Årzén

Researchers in the computers and control field are becoming increasingly aware of the need for an integrated scientific and technological perspective on the role that computers play in control systems and that control can play in computer systems. This need is evidenced by recent advances in areas such as embedded systems, plantwide control systems, robotics, and middleware. Control engineers must master computer and software technologies to be able to build the systems of the future, and software engineers need to use control concepts to master the ever-increasing complexity of computing systems. The increased awareness and activities within the field are underscored by *IEEE Control Systems Magazine* in this special issue and in the February 2003 special issue on software-enabled control.

Sanz (ricardo.sanz@etsii.upm.es) is with the Universidad Politécnica de Madrid, Spain, and Årzén (karlerik@control.lth.se) is with Lund University, Sweden.



0272-1708/03/$17.00©2003IEEE

Complexity is increasing, and the limitations of classical methods for real-time systems engineering become apparent when faced with the embedded controls required for critical applications. Control systems are becoming very complex software applications. Most of the software aspects that induce complexity have their roots in requirements that are hard to meet in isolation and extremely difficult to attain in conjunction with other subsystems. Many software systems for control are or need to be:

- *Time critical*: They require high performance and/or are subject to hard real-time constraints.
- *Embedded*: They must execute on platforms with limited computing resources and interact with the external environment.
- *Fault tolerant*: They must maintain (some level of) performance under fault conditions.
- *Distributed*: Software components are often distributed on several interconnected computers.
- *Intelligent*: They may require the solution of ill-posed problems, requiring a substantial level of intelligent and autonomous behavior.
- *Large*: They consist of potentially millions of lines of code.
- *Integrated*: An application requires the integration of several subsystems into a single, cohesive unit.
- *Open*: They should be nonproprietary and capable of supporting third-party applications.
- *Heterogeneous*: They should be able to execute on a variety and mixture of computational platforms.

Ten years ago, research in control software focused on one or a couple of these aspects. Today, even the simplest controller is subject to several of these requirements. Consider, for example, embedded systems for plant automation. They comprise key components in both end-user products and production systems. Machine controllers, process controllers, PLCs, robot controllers, and servocontrollers for devices such as fixtures and welding equipment are well-known industrial examples. Such embedded systems are real-time systems, typically distributed/networked, and integrated in large assemblies supporting standardized interfaces for openness and interoperability.

Control systems have traditionally been relatively static systems; however, technological advances and market demands are rapidly changing the situation. The evolution from static to dynamic systems makes flexibility a key design attribute for future systems. Local component intelligence is increasing, and large distributed controllers are being developed as communities of interacting intelligent agents. The increased connectivity implied by the Internet and mobile device technology will have a major impact on control system architectures. Modern products are often based on component architectures using commercial off-the-shelf (COTS) elements as units. Standardization and use of open market technologies are current requirements in control systems. New languages and platforms such as Java, C#, and CORBA are promising increased ease of use, portability, and safety and contribute to making heterogeneous distributed control system platforms possible.

Control systems constitute an important subclass of embedded real-time systems. In most critical cases, the requirements for predictability are imposed by the fact that the real-time system is actually a control system. A software-based controller is part of a computerized feedback loop that imposes timing constraints on the software and the platforms. The traditional approach in the real-time community is to design the system so as to maximize the temporal determinism. In control applications, however, the temporal nondeterminism can be viewed as a disturbance or uncertainty, and the control system can be designed to be robust against variations or to actively compensate for them. This opens up the possibility for more dynamic system architectures where the control applications and the implementation platform negotiate online for access to shared resources, such as CPU time and communication bandwidth. In this approach, the control performance can be regarded as a quality-of-service parameter.

Control engineers must master computer and software technologies to be able to build the systems of the future.

The rapid development of COTS computing and communication platforms lacking stringent timing guarantees makes static system designs based on worst-case assumptions increasingly conservative. Research is needed on design and implementation techniques that allow dynamic run-time flexibility with respect to variations such as changes in workload and resource utilization patterns. In addition, we need to improve our understanding of how this dynamic flexibility may be combined with more traditional real-time system approaches based on static design methodologies. For example, how should event-driven execution be combined with prescheduled time-driven execution in embedded control systems?

The use of control-based approaches for the modeling, analysis, and design of embedded computer and communications systems is currently receiving increased attention from the real-time systems community as a promising foundation for controlling the uncertainty in large and complex real-time systems. Areas of growing interest include feedback architectures for adaptive real-time computing, theory for performance guarantees under uncertainty, integrated resource scheduling and feedback control, control-theoretical models of dynamic real-time systems, application of control theory for controlling timing behavior, and optimal, robust, or adaptive feedback control in real-time systems. The use of control has the potential to increase flexibility while preserving dependability and efficiency. For example, control techniques can be used to compensate for shortcomings and imperfections in the implementation platforms. Control approaches to resource allocation are especially interesting for distributed control systems. For example, a feedback scheduler can distribute the computing and communications resources in such a way that the global control performance is maximized. This is also an alternative approach to increase system dependability or achieve graceful degradation.

Software engineers need to use control concepts to master the ever-increasing complexity of computing systems.

For future industrial competitiveness, new types of competence and system solutions are needed. The use of control-based approaches in the analysis and design of embedded systems is one promising approach. Furthermore, low-level, real-time technology needs to be combined with high-level aspects, such as programming, networking, security, simulation, and control.

The use of these technologies in the implementation of complex controllers will necessarily be based on development tools and methodologies that provide support for design, implementation, verification, and deployment. Real-time Unified Modeling Language (UML, the modeling language of reference in the software engineering community) will potentially provide us with methods for modeling real-time embedded systems in such a way that we will be able to compute real-time system properties (e.g., responsiveness, schedulability, resource requirements) in advance from the models.

When complexity increases, engineers rely on well-known, effective designs. Design knowledge capture is a critical issue for control engineers, who repeatedly re-use control designs that are well known and well documented. When software is involved in the final implementation, however, many other factors that are not well documented in the control design textbook must be taken into account. It is difficult to document the control and the software parts of a controller in a concise, coherent, and integrated way. An interesting methodology that can help us with this knowledge capture task is the use of design patterns for the systems in our field. These patterns will contribute to the effective sharing of the best design knowledge and will serve as a basis for effective systems development based on automated tool support.

To summarize, software is critical for the control systems community, and a better understanding of it is necessary to build the systems of the future in a framework of progressively difficult requirements: smaller size, less cost, less time, more functionality, more evolvability, and more dependability. To cope with these requirements, the control systems community must actively follow the developments in a number of areas: agent technology, architecture-based design, artificial intelligence, concurrent engineering, composability, design patterns, distributed embedded systems, domain engineering, embedded systems, frameworks, integration, life cycles, model-based software engineering, modular systems, object-oriented programming, ontologies, product line engineering, real-time distributed systems, reusability, software components, and software processes. This special section includes five articles that cover a part of this spectrum of knowledge.

Tools that unify the analysis and design of controllers and computing systems are critical for this field. The article "How Does Control Timing Affect Performance?" by Anton Cervin, Dan Henriksson, Bo Lincoln, Johan Eker, and Karl-Erik Årzén discusses the possibilities for analyzing the control performance effects caused by nondeterministic jitter (e.g., in sampling intervals and latencies) due to real-time kernel anomalies, communication delays, and the like. Two new MATLAB toolboxes are presented: Jitterbug and TrueTime. Jitterbug makes it possible to compute a quadratic performance criterion for a linear control system under various timing conditions. Using the toolbox, one can determine how sensitive a control system is to delay, jitter, lost samples, aborted computations, and so on. The tool can also be used to investigate jitter-compensating, aperiodic, and multirate controllers. TrueTime allows the user to simulate the timing effects

caused by scheduling, context switches, and interrupt handling in an RT kernel, as well as their impact on control performance. Additionally, the effects caused by various communication protocols on the performance of networked control loops can be simulated. TrueTime also serves as a simulation platform for the design of more flexible/adaptive scheduling strategies that take the application characteristics into account dynamically. One such example is feedback scheduling. The article primarily considers feedback scheduling in the context of control-loop scheduling (i.e., how to apply feedback scheduling ideas in the design of real-time feedback control systems).

Good design engineering is critically based on our competence in building good models of the systems we intend to build. In their article, Cervin et al. took an approach based on classic tools in the controls field; in "Using Models in Real-Time Software Design," Bran Selic and Leo Motus proceed along the same lines but from a different direction: that of mainstream software technology. Software engineering is critically conditioned by our capability to convey designs among all stakeholders in a complex software system (as most control systems are). This is especially critical when difficult requirements must be met (e.g., small footprint, speed, predictability, distribution). As the authors say, "real-time software is particularly difficult to design," but traditional model-based engineering is useful for this task. Multiperspective software models help designers focus on relevant details and ignore what is irrelevant for a particular task (analysis, understanding, code generation, etc.). Selic and Motus highlight the use of UML to model the real-time software implementation of controllers.

Transfer and storage of design knowledge is critical for systems engineering effectiveness. The article "Pattern-Based Control Systems Engineering" by Ricardo Sanz and Janusz Zalewsky introduces a pattern-based methodology for capturing control systems design knowledge. Patterns offer such a method for design knowledge transfer and are even more critical in domains with tight requirements, such as the control domain. When design heterogeneity grows, as is the case when control is augmented with complex software issues, it is progressively difficult to capture and interchange design knowledge. Patterns help in sharing knowledge and in the development of a common vocabulary and a shared understanding of design alternatives.

The increasing availability of distributed computing resources and the escalating need to fulfill complex behavioral requirements in the presence of uncertainty necessarily lead to the allocation of intelligence to the smallest parts of a control system. The article "Agent-Based Control Systems" by Nicholas R. Jennings and Stefan Bussmann describes the implementation of control systems composed of collections of semiautonomous agents that cooperate to reach a global control objective. The "agent" design paradigm offers a solid, generic foundation for control systems engineering. Encapsulating behavior in agents helps solve the composability problem of large-scale controller engineering, increasing functionality and systems resilience due to the increased autonomy of isolated entities.

The article "Feedback Performance Control in Software Services" by Tarek F. Abdelzaher, John A. Stankovic, Chenyang Lu, Ronghua Zhang, and Ying Lu looks at the other side of the coin. Software is critical for control systems, but control systems are also going to be critical for complex software systems. The authors describe ways of modeling the software system and give examples of how control technology can help increase the sustained performance of the system. They introduce control-theoretic concepts for the implementation of software systems that provide increased quality of service, and they describe tools and middleware platforms that can be used to regulate system performance attributes.

This special issue provides a sample of an extremely broad and complex field. Although we do not cover all the technologies and knowledge needed for a future synthesis of control and software, we hope that the sampling presented can help pave the way to a broader and more integrated knowledge sharing between both disciplines.

Ricardo Sanz is an associate professor of systems engineering and automatic control at the Universidad Politécnica de Madrid, Spain. He obtained his engineering and Ph.D. degrees from the same university in 1987 and 1991, respectively. He is a Senior Member of the IEEE and chair of the IFAC Technical Committee on Computers and Control and the OMG Control Systems Working Group. His research interests include large-scale controller engineering, intelligent modular control, and object-based distributed real-time systems. He is an associate editor of *IEEE Control Systems Magazine*.

Karl-Erik Årzén received a Ph.D. in automatic control from the Lund Institute of Technology, Sweden, in 1987. He has been a professor at the Department of Automatic Control at Lund Institute of Technology since 2000. His research interests are real-time systems, real-time control, and programming languages for control applications. He was chair of the IEEE Control Systems Society Technical Committee on Real-Time Control, Computing, and Signal Processing from 1999 to 2002 and is currently vice-chair of the IFAC Technical Committee on Computers and Control.