This theme issue includes three articles that explore various aspects of engineering complex systems that involve software. As these systems increase in complexity, so too must the design principles employed.
Engineering complex systems requires an extensive technical and semantic knowledge. Because of their complexity, components of these systems can interact in unpredictable ways—sequentially and concurrently. Consider, for example, a cyber-physical system (CPS) such as an electric smart grid that manages household solar generation and batteries for an entire neighborhood. Each home generates its own energy and either stores it for later use or sells it to a neighbor. How do we build such a system?

Let’s say we want to do this just among neighbors in a fog environment. The system needs to coordinate power electronics devices that can regulate power flow to and from each house. A negotiation needs to take place among distributed edge devices to decide a price and quantity. The impact on the power system interconnection should also be taken into account, and represented within the cyber system. If everything is negotiated and the impact accounted for, then energy can be discharged from one house and used to power another, creating an “energy Internet.”

What if a driver arrives home and wants to charge his electronic vehicle right away? If there’s enough energy stored or generated through the solar panels, then it isn’t a problem. But what about the second car? Is there enough? As more and more cars return home, how will the system prioritize which car gets the available energy, and how do we avoid overloading the power distribution system? Assuming we can navigate through those challenges, what about failures of the system and or attacks? How will these be mitigated?

There are more commonalities among smart grid, smart transportation, water purification, chemical plant, and medical/health systems than there are differences. Recently, NIST developed a CPS Framework, which is a model of the necessary elements for designing a complex CPS that alleviates all the re-thinking that usually goes on in complex system design. Aspects of control, timing, security, and resilience are all treated. It will be interesting to see how these efforts evolve in system design.

**IN THIS ISSUE**

The three papers in this theme issue on software engineering use different approaches for managing the design of complex engineered systems.

In “Wise Computing: Towards Endowing System Development with Proactive Wisdom,” David Harel, Guy Katz, Rami Marely, and Assaf Marron present a bold vision that gets to the heart of the complexity of system engineering by proposing computational design help. Instead of making the engineer think of all possible interactions, could a system “suggest” design alternatives and constraints? This visionary approach is not as far-fetched as it might seem; the world has common formalisms (such as ontologies that capture system knowledge) and languages to interact with them (through systems such as UML), not to mention reasoning engines. The challenge is to bring these elements together to help reason about both correct and incorrect potential executions.

In “Maintaining Consistency across Engineering Artifacts,” Alexander Egyed, Klaus Zeman, Peter Hegenberger, and Andreas Demuth give us a more straightforward approach for detecting inconsistencies in the design of systems, using UML as a representation. The semantic interaction of the objects within the system becomes key to capture, and OCL is proposed to represent these semantics. The use of this approach goes beyond consistency checking at design time, and it can be effectively applied at runtime to monitor for errors and security intrusions.

In the final paper, “Data Sharing Defined—Really!,” Ivan Handler explores the complexities of information sharing using his experience as a chief information officer in the medical field as a motivating example. Data at rest and data in motion are two key concepts that help to protect data from leakage and attack. Assuming that these methods reliably protect data, then the semantics of interaction become the primary concern: How do we share information among multiple, disjoint domains? Handler covers five distinct but interrelated aspects: privacy, reliability, interoperability, security, and trust—collectively referred to as PRIST.
The three papers in this theme issue have the potential to address the design challenges of complex systems spanning multiple domains and interacting in potentially unexpected ways. The knowledge required goes beyond basic computing concepts and design, to include an examination of interactions and multiple aspects of the application domain. We hope you find these articles interesting and insightful.

REFERENCES

ABOUT THE AUTHOR
BRUCE MCMILLIN is a professor of computer science at the Missouri University of Science and Technology. His research interests include cyber-physical security, distributed systems, and formal methods in software engineering. McMillin received a PhD in computer science from Michigan State University. He is a Senior Member of the IEEE Computer Society and serves on its Board of Governors. Contact him at ff@mst.edu.

IEEE Security & Privacy magazine provides articles with both a practical and research bent by the top thinkers in the field.

• stay current on the latest security tools and theories and gain invaluable practical and research knowledge,
• learn more about the latest techniques and cutting-edge technology, and
• discover case studies, tutorials, columns, and in-depth interviews and podcasts for the information security industry.