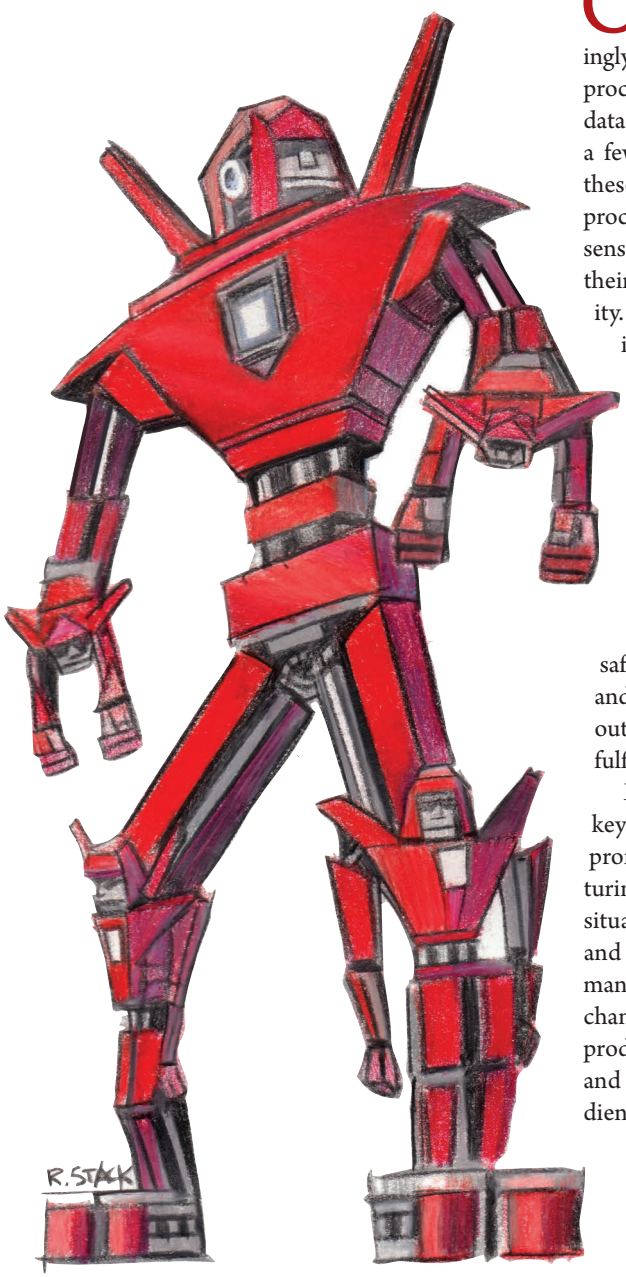
Editors: Mohamed Kaâniche, mohamed.kaaniche@laas.fr | Aad van Moorsel, aad.vanmoorsel@ncl.ac.uk

The Dependable Systems-of-Systems Design Challenge

Boudewijn R. Haverkort | University of Twente



Over the past decade, embedded systems have increasingly used data sensing and processing. Previously, embedded data processing dealt with merely a few on-off control signals. Now, these systems are capturing and processing much more complex sensory data and using it to improve their performance and dependability. In a recent discussion I was involved in, someone stated that "a 10-km car drive these days produces and uses more data than NASA used to put Armstrong on the moon." Manufacturers put all this ICT (information and communications technology) in cars to increase their performance, reliability, safety, efficiency, and comfort, and to decrease pollution. Without embedded ICT, cars couldn't fulfill EU exhaust regulations.

Embedded ICT also plays a key role in healthcare, energy, professional printing, manufacturing, distribution and logistics, situational awareness, avionics, and defense.[1] In all these sectors, manufacturers face dramatic changes in how they develop their products. In the past, "steel, oil, and rubber" were the main ingredients; now, ICT is increasingly determining product functionality, performance, dependability, and competitiveness. Almost invisibly, many traditional high-tech companies have over the years become ICT companies. The only difference from classic ICT companies such as Microsoft, Oracle, or SAP is the user interfaces.

As embedded systems evolve into systems of systems (SoSs), ICT's importance increases. On top of that, SoSs' characteristics make designing dependable SoSs even more demanding.

Embedded Systems

You could argue that the automotive systems I mentioned are "just" embedded systems. Indeed, people typically view an embedded system as a computer system (hardware and software) that's designed to interact with the physical world and that's part of a complete system including sensors and actuators. In short, they think of systems you buy in a box. The examples I just gave are of that type.

However, if we move to infrastructural systems such as those for surveillance or traffic control, the systems become much more geographically distributed and might change their structure and functionality over time. Even though such systems share many characteristics with embedded systems, they are more than that; that is, we're entering the SoS realm.

Systems of Systems

What makes SoSs different? Let's go back to the original description. An SoS is an assemblage of components

Copublished by the IEEE Computer and Reliability Societies

that are individually systems, with two additional properties:

- *Operational independence*. Disassembled components must be able to do useful work independently.
- *Managerial independence*. Disassembled components can work independently.[2]

Component systems also can have different owners and be subject to different legislation.

This is more complex than ordinary embedded systems, for various reasons. An SoS is an assemblage of systems, integrated out of independent components as they are, take it or leave it. That is, when you integrate the subsystems, you can't in any reasonable way make changes to them. Furthermore, integration might require runtime adaptation of components because some components might have been updated, thus requiring adaptation of the surrounding subsystems. This necessitates online integration and testing, normally only part of the offline design process. Normal integration and testing are already difficult and time-consuming, so this poses an extra challenge. In addition, the black-box nature of the components is high.

So, integrating the components in a way that guarantees extrafunctional properties such as dependability or performance is extremely difficult. Plus, security issues arise. Approaches involving service-level agreements, as in some networking or cloud-computing solutions, seem appropriate here. But many SoSs, unlike most Internet applications, involve applications with true real-time characteristics, making such approaches more challenging.

Three Examples

The following examples illustrate the promise and particular challenges of SoSs.

The Internet. Probably the best-known SoS is the Internet, in which many independent ISPs cooperatively provide worldwide connectivity, on the basis of jointly agreed-upon interfaces and protocols.[3] The Internet as we know it has been developed since the beginning of the 1980s, without any notion of SoSs.

Almost invisibly, many traditional high-tech companies have over the years become ICT companies. The only difference from classic ICT companies is the user interfaces.

Within an ISP's domain, that ISP has the freedom to choose its own implementation to some extent—for instance, for routing—as long as it adheres to the externally agreed-upon service levels and interfaces. However, the Internet at that level (the network layer) is a best-effort network—that is, a system that doesn't fulfill real-time requirements. We can learn lessons from the Internet, but that won't be sufficient for SoSs.

Cooperative adaptive cruise control. An example of a cooperative adaptive cruise control (CACC) system is the one in the Dutch Connect & Drive project[4] (see the video at www.youtube.com/watch?v=OoRuE7OqFEs). In a CACC-equipped car, the cruise control receives not only the usual internal signals (setpoint and car sensor readings) but also extravehicular signals (for example, radar or infrared signals measuring the distance between cars). In particular, Connect & Drive exchanged information using a wireless LAN connecting nearby cars. By doing this, cars could easily exchange information (digitally) about their position (on the basis of GPS readings), speed, and acceleration. Forwarding information from cars in front to cars behind lets cars look farther ahead than is possible with just radar or infrared communication. With such a system, cars could also communicate with roadside information stations or even subscribe to services providing detailed information about traffic situations.

In any case, the information received should be trustworthy and current; otherwise, basing speed adaptation on it poses a safety risk. In addition, these information channels won't always be active (owing to failures or other causes beyond any single car's control). So, a key requirement is robustness—the ability to deal with unplanned or undesirable circumstances.

Cars are typically privately owned and will be of different brands. GPS is provided publicly, as are roadside electronic information systems. Traffic services are typically commercially provided, using mobile phone channels by one or more providers. The overall system can provide, as an emerging service, much better, smoother traffic flow. However, CACC systems comprise differently owned components, and different legislation might be involved. Who's actually at the steering wheel? For accidents due to receiving incorrect information through one of the data channels, liability questions will arise.

Situational awareness. An example involving situational awareness is the Poseidon project.[5] Poseidon aimed to develop a system that combined information sources to give the Netherlands Coastguard an integrated view of coastal safety. This view could be of use for maritime collision avoidance,

search-and-rescue operations, maritime pollution response, and long-range detection and tracking of objects. The information to be integrated included real-time measurements (radio, radar, sonar, and satellite signals) and public and private databases of vessel locations (for example, see www.fleetmon.com or www.vesselfinder.com).

In such systems, the data available won't always be the same. Not all sources provide information at all times or about all vessels. The data formats aren't known a priori and might change over time. The data itself might be of varying quality and trustworthiness. Also, someone might try to inject incorrect information into the databases or send incorrect data though the system's communication channels. The challenge is to build a robust, adaptable system, without a predetermined configuration, that supports outlier detection. Clearly, each system that's integrated provides an independent functionality, but through cooperation, the combined system can function better and more reliably.

Changing Business

In the previous examples, the challenge to the manufacturer isn't just to make the initial physical product. It's about embedding ICT, about systems adapting over time, and about data mining and information processing. From the manufacturer's perspective, the focus shifts from selling products to providing services for sold products. This isn't new to high-tech industries: more companies focus on the services they provide than on the actual sales of their products. Examples are the telecommunications and printing-and-copying sectors. Even a company such as Rolls Royce is focusing on services for some of its aircraft engines. Already in the 1960s, they had introduced their Power by the Hour concept. However, many companies aren't prepared for this transition.

Toward Dependability

From what I just described, future SoSs will

- be large,
- be heterogeneous,
- have partly unknown subsystems (from the viewpoint of other subsystems),
- possibly vary their structure and cooperation over time,
- contain complex data dependencies, and
- have subsystems that must be able to communicate with a changing set of partners.

They'll also often have stringent performance and dependability requirements. Can we give the necessary guarantees?

Model-Driven Design

I advocate a model-driven-design approach for SoSs. Such an approach must be able to deal with the characteristics I just described. In addition, owing to SoSs' dynamic behavior, such an approach will need to adequately model notions of online design and integration (for additional components coming in or components that will be exchanged with older ones). These requirements are challenging, especially in light of current modeling and analysis tools and techniques, which typically don't scale well and usually require model homogeneity, time invariance, no data dependencies, and static structures.

Design Guidelines

Here are six guidelines for making a model-driven-design approach valuable for designing dependable SoSs. Each of these guidelines could be the starting point for many challenging PhD projects.

First, I don't expect a "single model class" to be possible or useful. Instead, I believe that cooperative modeling, through well-defined interfaces, is the best way to support design processes. An example of this approach is the European Destecs project (Design Support and Tooling for Embedded Control Software; www.destecs.org), which is combining discrete-event models with continuous (control) models. Approaches along these lines also allow for model inhomogeneity, so different design themes (disciplines) can employ their chosen methods.

Second, data dependencies' importance in SoSs means that we can't expect fully analytical model solutions. Instead, approaches that allow for hybrid simulation appear most fruitful.

Third, you can deal with uncertainty—for instance, about model parameters and other model components—in several ways, such as probabilistically, stochastically, or by using nondeterministic models. Researchers have made good progress with probabilistic and stochastic models. However, nondeterminism isn't compatible with simulation techniques. To investigate parameter uncertainties' impact, you can employ uncertainty and sensitivity analysis.

Fourth, if you don't know a subsystem's structure and behavior, approaches based on model mining and test-based modeling can produce overall behavioral models.

Fifth, compositional modeling and analysis are strong techniques; however, they require enhancement to deal with extrafunctional system characteristics such as dependability and performance. One such approach is the flow-equivalent server center analysis developed for computer performance analysis.[6]

Finally, modeling and analysis techniques must work with the state-of-the-art design tools used in industry. Industrial-system design engineers won't acquire and adapt to academically developed tools and techniques unless

they're embedded in the company-prescribed design flow.

The field of SoS design appears to be an excellent opportunity for computer scientists to team up with system designers and employ systems-engineering approaches from, for example, the aeronautics or automotive field. We shouldn't shy away from these approaches as being imprecise or too engineering-like; they put humans on the moon! Knowledge of classic research from these fields might actually help unleash the potential of the powerful techniques and tools that computer scientists have developed over the past decades. ■

Acknowledgments

This article was inspired by my experiences as scientific director and chairman of the Embedded Systems Institute (ESI) from 2009 to 2012. The opinions expressed are mine and not necessarily those of ESI or the companies ESI is or has been cooperating with. I gave presentations along the lines of this article at the First Workshop on Advances in Systems of Systems and at various UK and German universities in spring 2013. A paper will appear in *Electronic Proceedings in Theoretical Computer Science*.

References

1. K. Zimmermann and H.M. Hauser, "The Growing Importance of Embedded Software: Managing Hybrid Hardware-Software Business," Boston Consulting Group, 2004; www.bcgperspectives.com/content/articles/technology_software_growing_importance_of_embedded_software.
2. M.W. Maier, "Architecting Principles for Systems-of-Systems," *System Eng. J.*, vol. 1, no. 4, 1998, pp. 267–284.
3. J.F. Kurose and K.W. Ross, *Computer Networking: A Top-Down Approach*, 6th ed., Pearson, 2012.
4. T. Ploeg, A.F.A. Serrarens, and G.J. Heijenk, "Connect & Drive: Design and Evaluation of Cooperative Adaptive Cruise Control for Congestion Reduction," *J. Modern Transportation*, vol. 19, no. 3, 2011, pp. 207–213.
5. P. van der Laar, J. Tretmans, and M. Borth, *Situation Awareness with Systems of Systems*, Springer, 2013.
6. K.M. Chandy, U. Herzog, and L. Woo, "Parametric Analysis of Queuing Networks," *IBM J. Research and Development*, Jan. 1975, pp. 36–42.

Boudewijn R. Haverkort is a professor in Design and Analysis of Communication Systems at the University of Twente. Contact him at b.r.h.m.haverkort@utwente.nl.