A Software Architecture for Next-Generation Cyber-Physical Systems

Richard West and Gabriel Parmer

{richwest,gabep1}@cs.bu.edu Tel: +1-617-353-2065

1. Introduction

Next-generation cyber-physical systems (CPS) will pose great challenges in software system design, due to several key factors: (1) the immense diversity of hardware platforms, upon which distributed and embedded applications are to be implemented, and (2) the diversity of applications themselves and their requirements. Application requirements will not only have real-time constraints but will also encompass safety, security and dependability factors. No longer will simply designing a system around worst-case execution times and maximizing resource utilization be of key importance. While such considerations will be significant, the primary focus of system design will be to ensure reliable, safe, efficient and predictable behavior of the supported applications. This is evident from the continuing increase in software complexity, with tens of millions of lines of code predicted in the near future on embedded devices ranging from mobile phones to automotive systems.

The diversity of hardware platforms will greatly impact the design of system software, from the basic service abstractions to the scheduling, communication and coordination between services and applications. Moreover, judicious use of hardware features will be essential to ensure efficient resource usage (e.g., memory, battery power, CPU cycles, and communication bandwidth), as well as guaranteeing the necessary levels of reliability and safety/security.

Unfortunately, much of today's systems research is focused on the engineering and extensibility of existing commercial off-the-shelf (COTS) systems, to bridge the "semantic gap" between the needs of individual applications and the service provisions of the system. A significant effort is placed on circumventing the limitations of various system structures, APIs and the generic services designed in an application-agnostic manner. For example, monolithic systems such as Linux can be extended to allow untrusted users to incorporate their own service policies (e.g., for CPU scheduling that explicitly considers application deadlines) within the privileged protection domain of the core kernel. However, this may compromise the integrity and, hence, behavior of both the system itself and all resident applications. Consequently, a huge effort has therefore been placed on dealing with memory safety using hardware features such as paging and segmentation, and software techniques such as type-safe languages and software-based fault isolation. Other system structures, such as micro-kernels, isolate application-specific services outside the core kernel, but much of the research effort on the design of these systems has focused on efficient techniques to reduce the inherent communication costs imposed by service isolation. Fundamentally, a significant body of work has attempted to address the deficiencies of various system designs due to the mismatch between their service provisions and application needs.

Given the diverse application requirements and hardware features of future CPS, one system design solution will not suffice. Rather, what we need is a base system software architecture upon which services can easily be designed, deployed and composed on demand by individual applications, in a manner that satisfies specific safety, security, reliability, efficiency and predictability requirements, while still remaining within the bounds of given hardware capabilities.

2. Future Software Support for Cyber-Physical Systems

The vision is to design a system as a collection of application-specific services and abstractions, that are automatically structured and deployed on a target platform according to constraints in terms of: (a) hardware capabilities, and (b) application requirements. This differs from the view that one structure is always undeniably superior to another. Rather, this vision recognizes that the optimal system structure is contingent upon hardware features and application needs. Moreover, the system should be automatically composed with those services, and only those services, needed for the task at hand. For example, features such as a disk-based file management subsystem might be irrelevant to an embedded application (e.g., in avionics, automotive systems and future home appliances). In traditional systems, the burden of unneeded services has impacted resources such as memory and cache footprints, as well as code complexity. The consequences thereof can lead to safety and predictability violations, rendering reasoning about the behavior of a given system intractable.

Bridging the Semantic Gap. In current systems, applications request services of a trusted kernel via a well-defined interface (e.g., using system calls). Such interfaces, or APIs, are usually defined at a level that satisfies the common needs of a broad spectrum of applications, but this makes it awkward for applications to specify exactly what behavior they want from the underlying system. Future cyber-physical systems should enable applications to be programmed using the most appropriate system interface. By allowing applications to precisely specify their service and resource requirements, the system can transparently self-organize into the most beneficial structure, using the most appropriate services. For example, a carefully designed API may allow an avionics application to specify service requests for altitude and velocity control, so that specific height and air-speed requirements are met. Transparently, the system composes a series of lower-level services (e.g., to affect rudder, wing-flap and engine behavior), that are organized according to both application requirements and hardware features.

A Proposed Software Architecture. The aforementioned views suggest a system should be structured as a collection of component services, which may be isolated (using hardware and/or software techniques) or combined into a single address space according to well-defined requirements. For example, if the underlying processing hardware has a memory management unit (MMU), a series of component services may be automatically mapped to separate hardware protection domains, if this degree of safety is an important requirement of the application. Such isolation incurs inter-service communication delays, so the system must automatically arrange itself with dynamically-created communication channels to satisfy trade-offs between latency and isolation requirements. Likewise, issues such as predictability, in which service invocations occur at well-defined points in time, may be considered by the system as it adapts to the best possible arrangement. In essence, a system configuration may resemble a microkernel, a monolith, a layered service hierarchy, a distributed virtual machine, a single shared address space with application tasks, or some combination depending on hardware capabilities and application requirements. The challenge is to derive the best configuration from a collection of high-level service descriptions written by system developers in a hardware-independent manner, whereby carefully selected component services for the specific application are automatically targeted for a given hardware platform. Effectively, the system developer is no longer burdened with hand-optimizing a system design for a specific hardware platform but only needs to focus on the services available to applications.

A cyber-physical system architecture would sensibly consist of a small executive capable of dynamic service composition and component service isolation. Such an executive will be empowered with features capable of locating, authenticating, retrieving and communicating with remote services, as it is

unlikely that all services for a distributed embedded application will be physically local. Appropriately designed communication protocols will therefore be necessary so that services deployed across potentially wide-area environments can be accessed and invoked by trusted clients, in keeping with bandwidth and delay constraints, as well as others such as security.

Due to hardware heterogeneity, a cyber-physical system may encompass multiple devices with dissimilar instruction set architectures (ISAs). Traditionally, remote service invocation has involved complex marshaling procedures to exchange data between platforms in an architecture-independent manner. Taking this further, a unified hardware-independent ISA would be desirable for distributed services. When a service is deployed on a target platform it is recompiled and possibly linked with verification, communication, protection and other wrapper code for a specific hardware instruction set. Though this is similar to the idea behind Java byte-code, mechanisms such as non-real-time virtual machines, just-in-time (JIT) compilation, and software-enforced type-safety are not appropriate in embedded systems with limited resources and predictability constraints. Because services are compiled to a device-dependent ISA on demand, optimizations that satisfy the constraints of both the hardware and the application are possible. For example, function "inlining" can be applied to memory-rich devices, but not necessarily on those with limited memory. While system developers will produce services compiled for a unified hardware-independent ISA, embedded devices themselves may have the capabilities to locally compile services for their specific architecture. Similarly, cross-compilation on relatively powerful hosts may be suitable for services associated with devices having insufficient resources to run their own target compilers.

3. Summary

In summary, the major limitations of today's cyber-physical systems, from a systems software perspective are as follows: (1) there is a mismatch between application needs and system service provisions due to inadequate APIs making it difficult for applications to precisely specify the service they desire, (2) system services tend to be application-agnostic, typically focusing on fairness and efficiency rather than timing, safety, security and reliability constraints, and (3) current systems are too inflexible to be easily extended with specific services for target applications.

The challenges faced by future systems support for CPS include: (1) the design of an underlying software architecture that organizes itself with the most appropriate methods of communication and isolation between services, (2) the automatic composition of services to satisfy application constraints, given underlying hardware limitations, and (3) careful design of APIs and consideration of hardware heterogeneity in the generation and verification of a software system for a given application.

4. Brief Biographies

Richard West received an MEng (1991) from the University of Newcastle-upon-Tyne, England, as well as both MS (1998) and PhD (2000) degrees in computer science from the Georgia Institute of Technology. He is currently an assistant professor in the Computer Science Department at Boston University, where his research interests include operating systems, real-time systems, distributed computing and QoS management. Gabriel Parmer is a PhD student at Boston University, working on topics related to operating systems (especially their structure, service composition and extensibility), real-time systems and resource management. He currently holds a BA degree (2003) from Boston University.