Software Architecture Challenges and Requirements for Transportation Cyber-Physical Systems

Richard West and Gabriel Parmer

{richwest,gabep1}@cs.bu.edu

1 Introduction

A new field of cyber-physical computing is now emerging, involving communication and processing of data exchanged between physical devices and systems. In the transportation cyber-physical system (CPS) domain, there are numerous challenges posed by avionics, automotive and rail applications. For example, in the rail industry, monitoring and communication services are becoming increasingly important, to avoid train collisions and to maintain the highest throughput possible. This challenge is exacerbated by the de-regulation of rail services in many countries, that renders passenger and freight operations open to competition from private companies. In situations where track lines are shared across companies, it becomes essential to maintain effective communication between different train operators. If each company were to operate without cooperation it might be difficult to determine congestion or accident locations, thereby compromising safety and efficiency. Likewise, in the avionics sector, it is becoming increasingly important in our crowded airspace to ensure real-time monitoring and communication between competing airlines to avoid potentially catastrophic collisions. A similar situation exists with the automotive industry, in which emerging GPS-based services could be used to relay accident and congestion information to other vehicles in the vicinity. In all cases, it makes sense to provide an open standard for deploying communication and monitoring services on behalf of otherwise independent transport companies. Moreover, such services need to operate with some degree of predictability and dependability. Real-time requirements need to be met and the functional correctness of services needs to maintained, otherwise adverse consequences may arise. We, therefore, argue for a dependable and predictable software architecture, that offers an open standard for third-party vendors to deploy application-specific services.

1.1 Example: Automotive Services and Challenges

Before describing our requirements for a new software architecture, let us consider one particular transport domain in more detail. Here, we focus on the automotive sector and discuss some of the possible services that may be commonplace in the near future. One such service may provide real-time weather or traffic updates, software upgrades, or on-demand city guides via Wi-Fi access points, that enable in-vehicle Internet capabilities. Other vehicles outside the range of a Wi-Fi access point could form ad-hoc networks with neighboring vehicles that support such cooperation, thereby providing an indirect route to the Internet. Moreover, we may wish to allow competing automakers to upload services onto other vehicles that follow a cooperative agreement. Such services may be used for vehicle-to-vehicle traffic reports, or emergency services. For example, one vehicle ($V1$) in the range of city cameras might process and forward visual information, about an accident that has just occurred, to any vehicle having registered and been granted service with $V1$. The degree to which information is processed by $V1$ before it is sent downstream to other vehicles depends upon the service agreement with those other vehicles. For example, pre-processed data may be exchanged between $V1$ and other vehicles manufactured by the same automaker, while raw data may be sent to vehicles associated with a different automaker than that of $V1$. In the latter case, processing of the raw data is performed on the destination vehicle.

In the above service scenario we have a *trade-off between communication and computation*, with possible adaptations in the degree of computation performed local to any one vehicle. The actual communication versus computation trade-off depends on factors such as latency requirements placed on the service itself (e.g., the need for emergency information to be relayed to police, fire and ambulance services as soon as possible), and on the demand for resources (e.g., the availability and amount of CPU and I/O bandwidth needed by the service request). Other considerations with the deployment of traffic, accident and emergency services include the need to prevent malicious spread of false information. Bottlenecks may occur if vehicles are re-routed into congestion hot-spots due to false information from a malicious driver who is trying to find a traffic-free route. Likewise, emergency service vehicles such as ambulances must avoid being dispatched to incorrect locations while real accidents are left unreported. Here, then, is the need to isolate untrusted services, to prevent them from negatively impacting the behavior of an entire cyber-physical system. Moreover, we may need to quarantine certain services until they are deemed sufficiently trustworthy for more widespread deployment.

As a further example, consider a service provided on behalf of a government agency, such as a department of motor vehicles (DMV), that requests access to specific vehicle data, including average fuel usage and CO_2 emissions. This information may be used to monitor the energy and environmental impacts of an automaker's range of vehicles. Care must be taken, however, to avoid releasing sensitive information about the driver of a given vehicle, or perhaps an individual vehicle identification number, except in cases where it is considered necessary (e.g., if police wish to identify a stolen vehicle). Once again, we have a situation that requires a specific level of trust between the service requester (here, the DMV) and the information provider (here, the vehicle on which the service is deployed). Appropriate levels of isolation are needed between the service itself and sensitive information that should not be directly accessible. It is worth noting that even if a service is trusted it may still yield incorrect results if it is not scheduled predictably. In the above example, we want to make sure that a service to report fuel efficiency and CO_2 levels provides an accurate representation of the vehicle's true performance. In this case, sampling and processing sensor data, such as fuel consumption or exhaust emissions, must be performed in real-time.

As a final scenario, consider an automotive service that provides information about the expected range of a vehicle before it needs re-fueling and/or recharging. In the case of emerging electric vehicles, we may wish to provide services to determine nearby recharging points, suggest routes to such points based on driving patterns (e.g., average speed, distance traveled, and average energy consumption), and raise warnings when the nearest recharging location is about to go outside the vehicle's expected range. As with prior examples, services of this kind require accurate real-time information monitoring. Miscalculating energy or fuel ranges and distances to nearest stations may lead to vehicles being stranded.

In summary, the above examples argue for an open software architecture, that is customizable with a diverse range of services. Third party providers of such services need to cooperate, even when there is the potential for mistrust and malicious behavior. Services may need to be deployed remotely as well as locally, to satisfy system-wide resource constraints. This raises a number of communication versus computation trade-offs, and leads to possible adaptations in the placement of services as well as their degree of isolation. Moreover, the extent to which services are isolated impacts their resource usage (e.g., CPU and network bandwidth), which in turn affects the end-to-end latency of inter-service communication. Sensor readings and corresponding actions need to be performed according to real-time constraints, with precise resource accountability. Groups of diverse services deployed on a given processing platform (e.g., within one vehicle) need to be scheduled according to appropriate deadlines. Likewise, cooperating services spanning an entire cyber-physical system need to be composable, so that end-to-end timing requirements can be met.

1.2 Software Architecture Requirements for Next-Generation CPSes

The requirements for a software architecture addressing the above challenges include:

- *a standardized operating system platform on which individual vendors can deploy application-specific services* – These services must be isolated in a manner that ensures software integrity and performance guarantees [4, 1]. Services must be composable, so that relatively basic software components can be combined to provide richer functionality. Interactions between composable services must nonetheless adhere to specific resource and performance requirements;
- *appropriate levels of service isolation, so that services are only granted the privileges necessary to accomplish their task* – This means that untrusted services, deployed by third-party vendors, are effectively sandboxed in such a way that their scope of impact on the system is limited [4];
- *predictable service execution* – In this case, mechanisms and policies are needed to guarantee real-time execution of services, and to ensure resource and performance isolation between separate services [3]. It follows that proper accountability of system resource usage is essential [5]. Additionally, trade-offs between the degree of isolation of separate services and the costs of inter-service communication must be considered, so that end-to-end resource requirements are met [2];
- *adaptability of system configurations according to changes in the physical characteristics of CPSes* – For example, dynamic changes to vehicles within a geographic region affect resource availability within the corresponding CPS. The system must adapt to accommodate these changes. Service functionality and isolation may need to be altered, and the placement of services within a system may need to be adapted as necessary.

2. 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 Associate Professor in the Computer Science Department at Boston University, where his research interests include operating systems, real-time systems, distributed computing and resource management. In particular, Professor West is working on current research goals that focus on the design of dependable and predictable computing systems. 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. Gabriel is currently developing the “Composite” component-based system, to support “mutable protection domains” and user-level configurable services. He currently holds a BA degree (2003) from Boston University.

References

[1] G. Parmer and R. West. Hijack: Taking control of COTS systems for real-time user-level services. In *Proceedings of the 13th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2007.

[2] G. Parmer and R. West. Mutable protection domains: Towards a component-based system for dependable and predictable computing. In *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS)*, Tucson Arizona, USA, December 2007.

[3] G. Parmer and R. West. Predictable interrupt management and scheduling in the Composite component-based system. In *Proceedings of the 29th IEEE Real-Time Systems Symposium (RTSS)*, December 2008.

[4] R. West and G. Parmer. Application-specific service technologies for commodity operating systems in real-time environments. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2006.

[5] Y. Zhang and R. West. Process-aware interrupt scheduling and accounting. In *Proceedings of the 27th IEEE Real-Time Systems Symposium (RTSS)*, December 2006.