System Virtualization Scoping (Green Hills Software)

Added by Michael Kerrisk, last edited by Youngwoo Park on Dec 16, 2011

Editing History

- Oct. 17, 2011 - David Kleidermacher, Green Hills Software – Initial Draft

Executive Overview

System virtualization is a growing OEM requirement in infotainment platforms due to two primary drivers:

- *ECU consolidation* to save cost, footprint, weight
- *Mixed criticality sandboxing*: strong isolation of critical functions from the general-purpose operating system (in this context, the GENIVI compliant Linux) and its functions. Critical functions include hard real-time, safety-critical, and/or security-critical applications that may not be suitable for Linux.

These are not independent concerns: consolidation of infotainment processors and other ECUs often forces a mixed criticality requirement. For example, a consolidated platform may include traditional center stack multimedia and navigation functionality as well as safety-critical rear-view video, real-time cluster, vehicular network communications, and security-critical telematics.

While system virtualization is well known in the computing world to provide numerous flexibility, portability, and efficiency benefits, emergence of system virtualization in embedded applications has been made practical only over the past few years due to commercialization of highly efficient embedded and mobile hypervisors, increased CPU horsepower, and hardware-assist features, such as Intel Virtualization Technology (Intel VT), ARM TrustZone, and ARM Virtualization Extensions (ARM VE).

The GENIVI Alliance would benefit from increased scope to include system virtualization because vehicle manufacturers and ecosystem suppliers would be able to take advantage of pre-integrated, GENIVI compliant virtualization solutions that provide the aforementioned benefits at significantly reduced time and cost relative to integration of proprietary solutions that lack standardized interfaces and independent interoperability validation.

The primary purpose of this document is to enumerate the major potential compliance attributes for system virtualization. The secondary purpose is to provide some guidance and recommendations with respect to the respective benefits and challenges for each of these major attributes. The intended audience of this document is GENIVI leadership for its use in deciding whether to expand GENIVI's scope to include system virtualization.

Scope of Compliance Attributes

Ability to host GENIVI compliant operating systems

Overview

A compliant hypervisor must be able to host GENIVI compliant infotainment operating systems (as *guests*). With GENIVI virtualization compliance, the burden of virtualization validation resides on the virtualization platform vendor and is less likely to adversely impact GENIVI operating system vendors nor its upstream users. Furthermore, the availability of GENIVI virtualization compliance validation enables OEMs and Tier-1s to rapidly adopt and deploy virtualization technology and apply it as a differentiator in their designs.

Questions/Issues

1. How many vendor solutions must be validated to achieve compliance?
    a. There is risk that a hypervisor vendor would validate a single implementation and that other vendor implementations would not work. That would defeat the compliance/standardization purpose.
    b. It is not practical to validate all possible GENIVI-compliant operating systems
2. For virtualization-related changes to GENIVI Linux, allow hypervisor vendor-maintained GENIVI Linux patches or modifications maintained by the GENIVI Linux vendor?

Recommendations

1. *REQUIRE* hosting of GENIVI compliant operating systems
2. *REQUIRE* compliance testing of at least two concurrently executing, heterogeneous implementations (i.e. GENIVI Linux from multiple vendors)
3. *REQUIRE* publication of validated GENIVI operating systems and ease of portability documentary evidence
4. *DO NOT REQUIRE* standardized/limited modifications to the GENIVI Linux. While it would be desirable to standardize or at least require GENIVI Linux vendor official support of virtualization-related patches, this requirement is not realistic given the current state of hardware in which CPU and I/O virtualization often requires significant, hypervisor vendor-specific modifications and the fact that there is no significant standardization of CPU and I/O virtualization interfaces between hosted OS and hypervisor. Furthermore, the ability to handle shared I/O is a differentiator amongst virtualization offerings.

Standardized communications between multiple guest GENIVI compliant operating systems

Overview

A compliant hypervisor must ensure that hosted guest GENIVI compliant infotainment operating systems are able to communicate in a useful and interoperable manner.

Questions/Issues

1. Communication mechanisms vary based on requirements. For example, one-way, read-only information exchange for a health monitor observing the guest may be desirable. Some systems may have stringent performance (throughput and/or latency) requirements; e.g. it may be required that a system-level message to disable the center stack volume be transmitted to a guest operating system within a known, deterministic amount of time. So the question is how broadly applicable should a compliant mechanism be?
2. Although IP is necessary mechanism for communication, we should check the overhead of IP stack. When GENIVI communicates to control domain (SW) on hypervisor, the performance (real time) issues could be more important. In addition, there are software partitions (guest OS) without IP stack and GENIVI need to communicate to them. If necessary, we need to consider the native APIs or standard protocols for communication. Youngwoo Park

Recommendations

1. *REQUIRE* at least one form of standard Linux inter-process communication: an Internet Protocol (IP) bridge between guests is arguably the most standardized possible way for two arbitrary guest operating systems to communicate. With an IP bridge, guests can communicate using practically any higher-level IP-based API or protocol, including network sockets, web services, CORBA, SNMP, etc. Compliance validation is simple: ensure that guests can communicate over IP (e.g. a simple UDP socket would suffice).
2. *DO NOT REQUIRE* specific mechanisms for configuring/establishing the local IP-based bridge (e.g. API, management interface, DHCP, NAT, etc.). While this may be desirable, it is not viewed as critical or complex and therefore not important for the first version of a compliance specification. The specific layer two mechanism for hosting the IP bridge is also not significant enough to standardize. Linux has multiple methods for installing custom IP interfaces, such as virtual Ethernet bridges and TUN interfaces, and virtualization solutions typically require nothing more complicated than installing a Linux kernel module to create the IP-based communications mechanism.
3. *DO NOT REQUIRE* any specific APIs or protocols. The downside to the IP bridge is that it implies data must traverse the Linux network stack. Many hypervisor vendors provide a higher performance and/or real-time communications mechanism to augment IP networking approaches; however, since these are not standardized elsewhere, compliance validation is unlikely to be sensible. We may want to consider an *OPTIONAL* communications mechanism that can provide higher-performance (throughput and/or latency).

I/O virtualization and sharing requirements

Overview

Arguably the most challenging aspect of system virtualization is the management of I/O peripherals across virtual machines. In particular, the sharing of a single physical peripheral, such as a graphics processing unit (GPU) or WiFi interface, across multiple virtual machines is sometimes required. Yet, current generations of common infotainment system applications processors provide little or no hardware assistance in this reliable (availability of service, secure partitioning of information flow, etc.) sharing. Thus, the ability of hypervisors to share various flavors of I/O is variable, and the guest OS modifications sometimes needed to affect sharing is often non-standardized.

Questions/Issues

1. Should GENIVI require some form of I/O sharing for common infotainment/ECU consolidation scenarios? One example would be sharing a single GPU between the center stack and a virtualized 3D cluster. Another example is an audio device between center stack, rear-view camera system, and rear-seat entertainment.
2. Linux does have some limited standardization of paravirtualized interfaces that permit sharing of block-oriented peripherals, such as network and file system devices. Should these be mandated/adopted?
3. In my opinion, it is possible to discuss some mandated sharing requirements for very essential I/O drivers like Ethernet. Youngwoo Park

Recommendations

1. *DO NOT REQUIRE* GENIVI compliance for I/O virtualization. This is simply too complicated a topic to address in an initial specification.

Ability to host AUTOSAR environment(s)

Overview

Some types of guest operating systems may have specific performance and/or latency requirements that may be challenging or impractical to meet depending on the system virtualization approach taken. Therefore, it is important

that GENIVI virtualization not preclude important system architectures. One obvious and increasingly important example is AUTOSAR. Executing an AUTOSAR OS in a system virtual machine is impractical in many virtualization environments. For example, the AUTOSAR OS may be unable to directly handle interrupts without interposition by a hypervisor. The combined overhead of the hypervisor interrupt handling and re-direction to the guest may result in loss of real-time responsiveness.

Questions/Issues

1. Does validating AUTOSAR capability in a virtualized environment imply or require first that GENIVI validate AUTOSAR operating systems?
2. AUTOSAR is only one form of real-time automotive ECU operating system. Legacy OSEK and T-Kernel are two more examples that may be important in some automobiles that also adopt GENIVI. Should compliance be specifically aimed at AUTOSAR, or should compliance aim to ensure that the virtualization environment is able to host hard real-time workloads alongside the GENIVI-compliant infotainment OS? Or should we omit real-time entirely?

Recommendations

1. *REQUIRE* that the hypervisor support hard real-time threads written in the C programming language. These threads may execute on a microkernel (if the virtualization is microkernel-based) or in a simple virtual machine if the virtualization solution is able to guarantee adequate worst-case latency for that virtual machine. Validation should include interrupt latency and/or response time when the CPU is also heavily loaded within at least one GENIVI-compliant infotainment OS. By validating real-time compliance, GENIVI would serve to ensure that virtualization approaches do not preclude the use of any ECU environment that can be adapted to run on the hypervisor, reducing risk for system developers.
2. *DO NOT REQUIRE* AUTOSAR-specific testing. This could be an *OPTIONAL* compliance item or perhaps a future addition to GENIVI compliance when consolidated infotainment and AUTOSAR applications are further in demand. In addition, it seems unreasonable to require AUTOSAR guest compliance when the AUTOSAR environments themselves are currently out-of-scope. Perhaps this could be reconciled by requiring the hosting of AUTOSAR environments from at least some subset of the leading independent vendors (Vector, Elektrobit, ETAS).

Standardized SDK/Configurator for hypervisor

Overview

When adopting virtualization in a system design, one of the most important issues that can slow development and integration is the programming and configuration of the hypervisor itself and how the system designer manages physical resources across virtual machines. It is desirable to aid system developers by standardizing this aspect of virtualization.

Questions/Issues

1. Unfortunately, there is no ubiquitous hypervisor configuration standard across all major CPU architectures. The best attempt at virtualization resource management and configuration standardization to date may be embodied by the power.org ePAPR specification. This specification documents how hypervisors must present resources to guest operating systems and how guest operating systems must locate these resource definitions and process them. ePAPR uses the elegant concept of a *resource tree* for this. The boot loader provides a system-wide resource tree for the hypervisor that in turn provides a resource tree for the virtualized resources presented to each guest. An ePAPR-compliant guest processes the memory-resident device tree in order to locate and then initialize its hardware resources. ePAPR-compliant hypervisors and Linux operating systems are developed and maintained by multiple power.org members. Unfortunately, power.org and its ePAPR

specification govern only Power Architecture and are not followed by Intel or ARM architecture ecosystems. Neither Intel nor ARM has similar standardization bodies that have attempted to specify virtualization infrastructure. It seems a bit unrealistic for GENIVI to attempt to bridge this gap.
2. One of the big burden for consolidation is configuration (hardware, device driver, middleware). Although standardized SDK is not necessary, hypervisor and GENIVI configuration could be interworked. (For example, unused device driver and middleware are removed from partition depending on hardware configuration). Youngwoo Park

Recommendations

1. *DO NOT REQUIRE* any form of standardized configuration or software development kit (SDK). If system virtualization becomes more widely adopted, GENIVI should consider adopting some form of configuration standardization, perhaps something as simple as an XML file format that is written only for the hypervisor and specifies a minimum required set of boot-time assignable (static) resources such as RAM allocation per guest. Additional configurable parameters could be specified in a standardized syntax but left out-of-scope semantically.

Local and remote management of hypervisor and virtual machines

Overview

Some types of embedded systems have important and increasing requirements for remote management of various sorts. The automobile is a good example: management connections can be used for remote diagnostics and maintenance, remote firmware/software updates, multimedia content uploads, etc. These capabilities can significantly reduce maintenance and user cost. While the GENIVI infotainment OS is an obvious candidate for direct management (e.g. multimedia content uploads, software/kernel updates), the virtualization layer is also an obvious candidate as exemplified by the management suites available in data center, server-based virtualization (VMware, Microsoft, Citrix). Out-of-band management provides some distinct advantages of managing within the infotainment OS. For example, out-of-band management enables improved availability – remote maintenance personnel can reboot the entire center stack, potentially avoiding a costly service station visit.

Questions/Issues

1. Should GENIVI first focus on remote management of the infotainment OS itself? This could be performed in-band by a Linux system application (for updating the root file system), in a boot-loader, or in the hypervisor.
2. Today, there is very little standardization in the world of remote management for hypervisors. In the data center, VMware, Microsoft, and Citrix all implement proprietary communication protocols and APIs. This is a headache for IT administrators and will be for automotive designers, but should GENIVI attempt to bridge this gap? Is there something simple that GENIVI can do to move this in the right direction and promote standardized remote management approaches?
3. I don’t know much about DASH. Does DASH 1.1 profile includes out-of-band management specification? In my thought, GENIVI needs to be implemented DASH profiles for out-of-band management. We should discuss the necessity of DASH. Right? Youngwoo Park

Recommendations

1. *REQUIRE* the use of the Distributed Management Task Force (DMTF) Desktop and Mobile Architecture for System Hardware (DASH) framework be used for remote management of GENIVI-compliant hypervisors. By reusing an existing open standard, GENIVI can benefit from an existing compliance validation program (DASH conformance program), existing third-party management products, and seamless integration into existing back-end management infrastructures. As of October 2011, there are over 50 products certified conformant to DASH.

2. *REQUIRE* that the DASH agent execute out-of-band from the primary infotainment OS so that the entire infotainment OS can be managed (e.g. patched, replaced, rebooted). The DASH agent can execute either as a native microkernel application (for microkernel-based hypervisors) or an application on an isolated guest operating system (virtual appliance).
3. *REQUIRE* implementation of the mandatory DASH 1.1 profiles, the starting point for DASH-based management infrastructures. These profiles are: Base Desktop and Mobile, Profile Registration, Role Based Authorization, and Simple Identity Management. GENIVI may want to consider *OPTIONAL* compliance to additional profiles, such as Boot Control, IP Interface, OS Status, and Software Update. Note that the proposal to adopt DASH compatibility is not intended to be prescriptive of management functionality, but rather as an open standards-based framework to facilitate interoperable solutions that can provide automotive designers with better management functionality faster than otherwise practical. It is expected that system designers will customize their management APIs and capabilities based on their internal needs and competitive differentiation.

Ability to meet ISO 26262 requirements (hypervisor and hosted applications)

Overview

Some types of consolidated infotainment/ECU applications may have specific safety requirements that may be challenging or impractical to meet depending on the system virtualization approach taken. Therefore, it is important that GENIVI virtualization not preclude important system architectures. One obvious and increasingly important example is ISO 26262, a functional safety standard that is enjoying rapid adoption in the automotive sector. Executing ISO 26262 compliant safety-critical applications in a system virtual machine is impractical in many virtualization environments. For example, ISO 26262 automotive safety lifecycle requirements may not be realizable by an open source development process used by Linux or Xen. Furthermore, the hypervisor may not be designed for safety-critical environments and lack the assurance evidence required to meet ISO 26262 certification. While ISO 26262 certification of a system may be possible without an ISO 26262 certified hypervisor, this is unproven and adds significant risk relative to a system whose hypervisor is ISO 26262 compliant and/or certified.

Questions/Issues

1. While ISO 26262 is the wave of the future, the standard is in the process of completion/publication, and certifications are just starting. Is it too much to ask for certification of the hypervisor? Is there something less than an official 26262 certification that would provide sufficient assurance to system designers that the hypervisor will not introduce system certification risk?
2. What automotive safety integrity level (ASIL) would be sensible to target? If the level is too low, then it may preclude the realization of higher system-level SIL requirements.

Recommendations

1. *DO NOT REQUIRE* hypervisor ISO 26262 certification. This can be considered later when the standard is more mature.
2. *REQUIRE* that the hypervisor be certifiable to ISO 26262 ASIL D. The definition of "certifiable" is open to discussion, but one approach would be to accept certification to its ancestral, more generalized, and more mature standard, IEC 61508, at an equivalent level (SIL 3). Another approach would be to have a written letter from a certification authority (TUV or Exida) that states that the hypervisor product meets 26262 development processes and is unlikely to have a showstopper in a certification process. Another approach would be to require CMMI Level 3+ or equivalent SPICE rating for the development organization that produces the hypervisor product. While CMMI does not imply safety, it implies a level of development organization maturity that is far more likely to be successful at achieving ISO 26262 compliance.

Labels: None