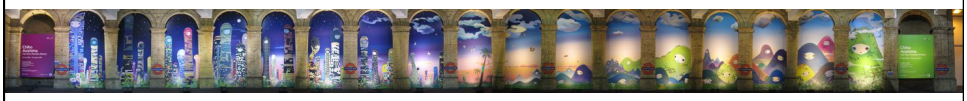
A Composition Kernel for Multi-core Dependable Embedded Systems



Tatsuo Nakajima, Yuki Kinebuchi
Ubiquitous and Distributed Computing Lab.
Department of Computer Science
Waseda University

IFIP WG 10.4 Meeting Jan 23 2010

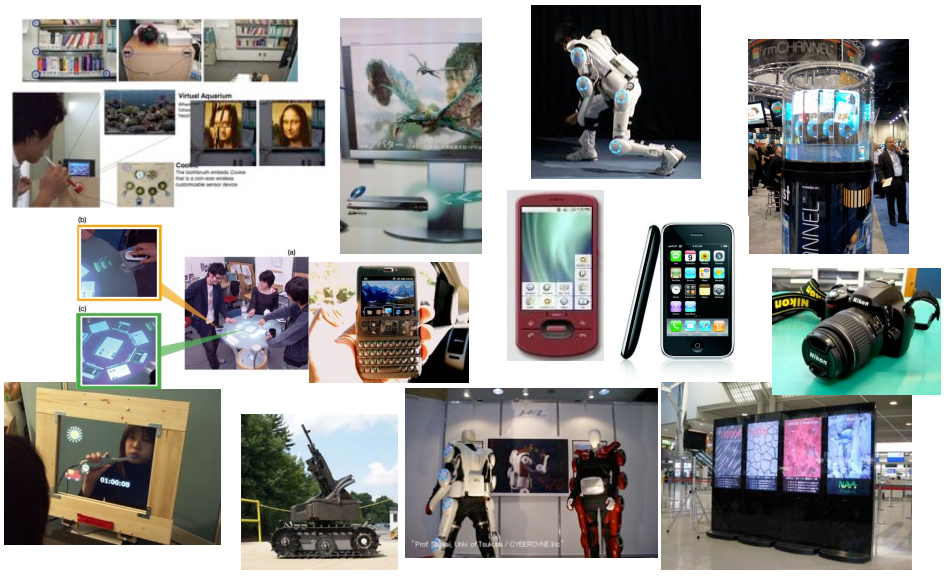
Overview

- Composing Multiple Functionalities.
- Overview of Our Research in the DEOS project
- Composition Kernel
  - Real-Time Resource Management
  - Dynamic Virtual CPU Management
  - Proactive Recovery Management
  - Integrity Management
- Conclusion

Composing Multiple Functionalities

- Reducing Cost, Complexity, Increasing Maintainability
  - Many dedicated hardware to integrated hardware
- Cloud Computing
  - Composing various information servers
    - Virtual machine monitor: VMWare, XEN
- Cyber-Physical Systems
  - Composing various control functions
    - ARINC 653(APEX)
- Information Appliances
  - Information Processing
    - Interaction with Human
  - Control Processing
    - Interaction with Physical Environment

Ambient Intelligence and Smart Product



From Usability to Persuasion

- Human decision is becoming harder due a large amount of low quality and less structured information.
  - If the environment returns appropriate feedback to users by showing the effect of the current behavior, the users' life become more dependable because they can think more rationally.
  - From entertainment to caring.

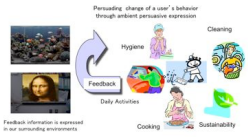
We are impossible to escape from various attractive temptation.

Integrated Architecture in Ambient Intelligence

Intelligent Robot

Sensor Network

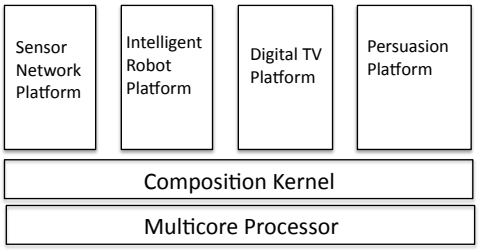
Digital TV



Persuasion Platform

Need to integrate information processing and control processing

- Low Overhead
- High Reliability
- Real-Time Support
- Supporting Multicore Processors
- Reusing applications and OSes with minimum modification



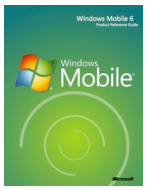
Open Systems and Composition

Intelligent Robot

Sensor Network

Digital TV

symbian OS



Sensor Network Platform | Intelligent Robot Platform | Digital TV Platform | Application Environment

Composition Kernel

Multicore Processor

Platform for Composability

DEOS Framework

DEOS Linux | SMP Support

RT App. | L4

RT App. | Toppers

Device Drivers | Thin Kernel

Integrity Management | Thin Kernel

Composition Kernel

Multicore Processors

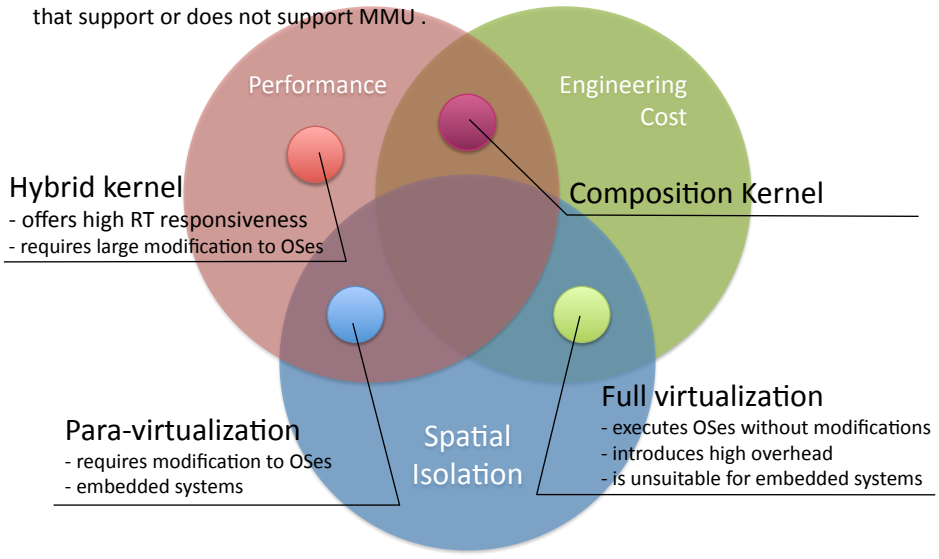
Composition kernel allows multiple OSes to run on multicore processors with the minimum modification.

Additional Value by Composition Kernel

- Increasing composability of the entire system.
  - Decreasing the cost to port existing applications without violating the timing constraints of RTOS.
- Using multicore processors in a more flexible way.
  - Reducing the number of cores according to the system load and handling dead cores.
- Offering more flexible rebooting/recovering strategies.
  - Enabling to use more optimistic strategies to increase the availability.
- Maintaining the integrity of OS kernels.
  - Detecting and repairing the violation of the integrity.

Design Issue

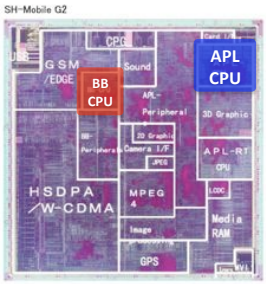
Our approach focuses on various 32-64 bits processors for embedded systems that support or does not support MMU .

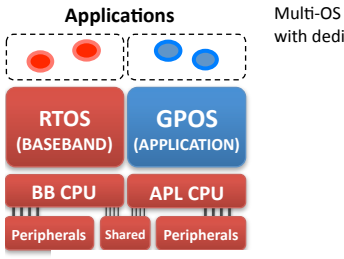


Dependable Composabiity

- Real-Time Virtual CPU Scheduling
  - Multiplexing CPU by multiple OSes
- Dynamic Multicore Virtual CPU Management
  - Migrating VCPU among cores.
- Proactive Recovery Management
  - Independent recovery of multiple OSes.
- Integrity Management
  - Repair the integrity of guest OSes.

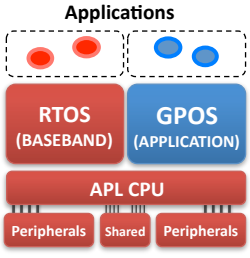
Real-Time Resource Management





Multi-OS platform
with dedicated processors

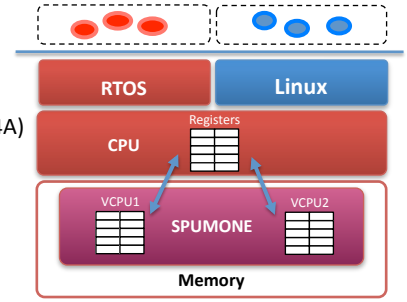
Multi-OS platform
With integrated processors



A Composition Kernel: SPUMONE

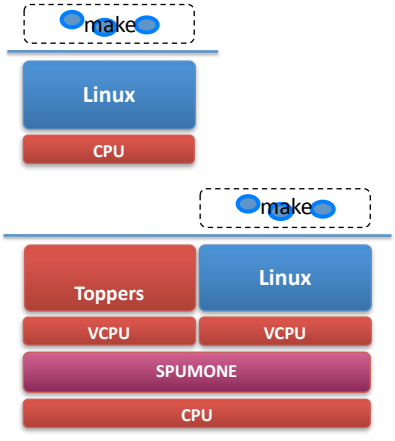
- OS kernels and SPUMONE reside in the same privileged address space.
- SPUMONE switches the execution of OSes by storing and loading the processor registers from and to the memory
- A main abstraction of SPUMONE is virtual CPUs(VCPU).
- CPU: SH7780(SH-4A ISA)

SH7780 400MHz (ISA: SH-4A)
2 serial channels
6 timer channels
128MB DDR-SDRAM
100Mbps Ethernet x 2
CF card adapter
2.5inch IDE adapter



Overhead Measurement

- Compared the time took to build a Linux kernel with native Linux and with virtualized Linux



Linux kernel build time

Configuration	Time	Overhead
Linux only	68m5.898s	-
Linux and TOPPERS on SPUMONE	69m3.091s	1.4%

- The overhead includes the time consumed by TOPPERS timer interrupt handler (1ms period)

RTOS: TOPPERS/JSP 1.3
kernel and applications reside in the same address space
64MB
Devices
Serial channel 0
Timer channel 3
GPOS: Linux 2.6.20.1
64MB
Devices: All the other
Root FS: NFS share (Fedora Core 5)

Engineering Cost

Modifications to guest Linux kernels (*.c, *.h, *.S, Makefile, Kconfig)

OS	Added LOC	Removed LOC
Linux 2.6.20.1 on SPUMONE (SH)	56	17
RTLinux 3.2 (Linux 2.6.9 / x86)	2798	1131
RTAI 3.6.2 (Linux 2.6.19 / x86)	5920	163
OK Linux (OKL4)(Linux 2.6.24 / x86)	28149	-

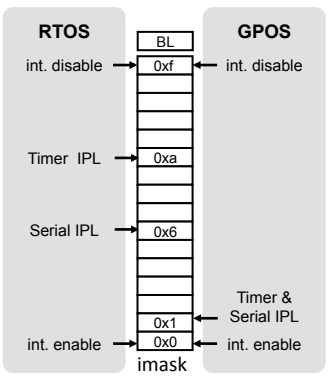
- We only compared the modified LOC of the Linux kernel
- The results do not include the LOC of the device drivers which provide communication channels between an RTOS and Linux

- `$ diffstat kerne_patch-2.6.9-rtl3.2-rc1 # removed 'defconfig's`
- `$ diffstat hal-linux-2.6.19-i386-1.7-01.patch`
- `$ ohcount asm-l4`

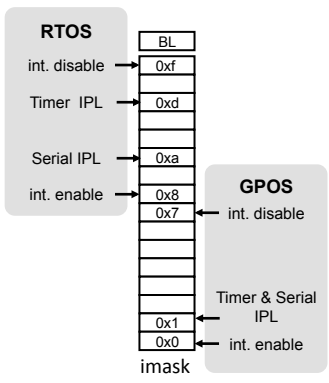
Virtualizing Interrupts

- Interrupt priority level assignment
  - Utilize the interrupt priority level (IPL) mechanism of the SH processor

IPL = Interrupt Priority Level

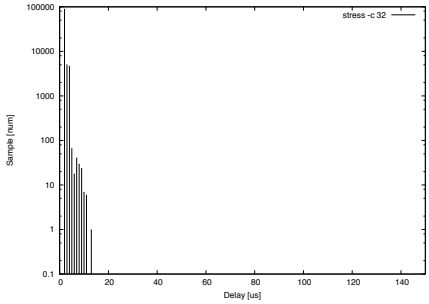


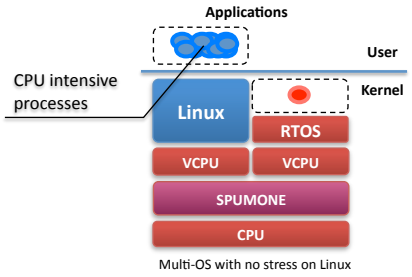
(a) Native OS



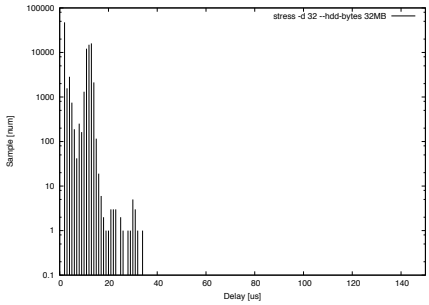
(b) Modified OS

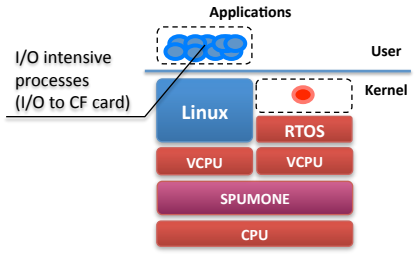
Temporal Isolation





Multi-OS with no stress on Linux

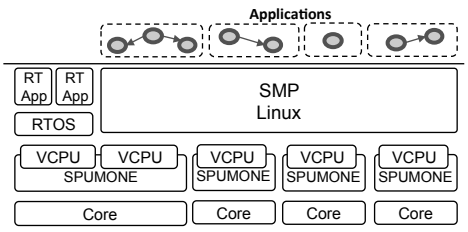




Multi-OS with I/O stress on Linux

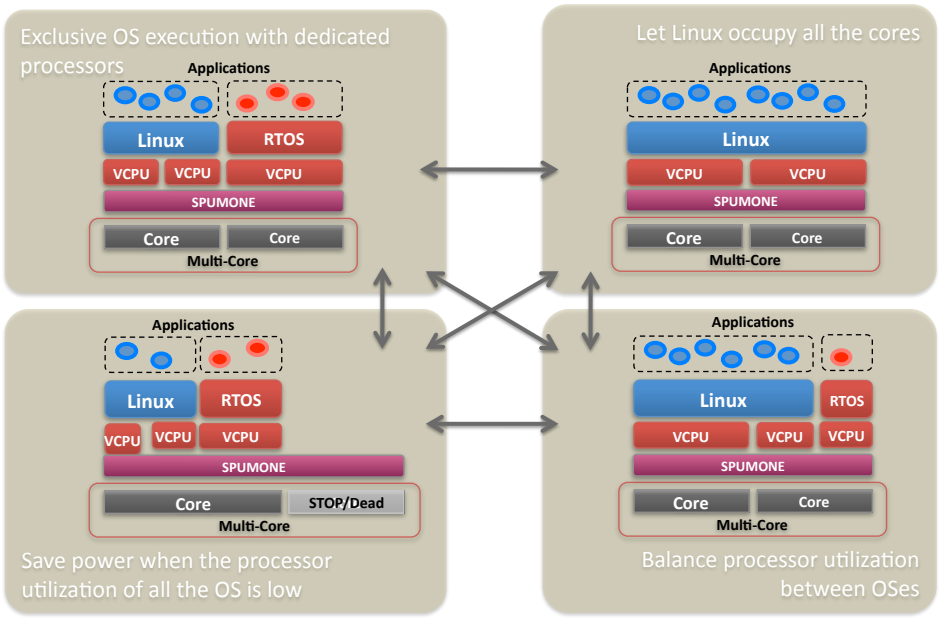
Dynamic Virtual CPU Management

- Distributed system model (Multikernel)
  - One SPUMONE on each processor
  - Each SPUMONE schedules VCPUs on its processor core.
  - Each SPUMONE communicates with message passing model implemented on inter-processor interrupt.
    - VCPU can be migrated by passing its state to another SPUMONE



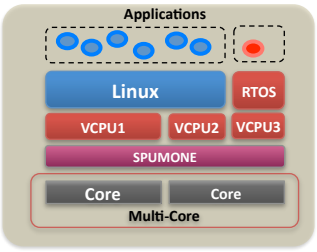
MSRP1BASE02
RP1 (SH-4A MP ISA)
600MHz x 4
128MB DDR-SDRAM

Dynamic Virtual CPU Management

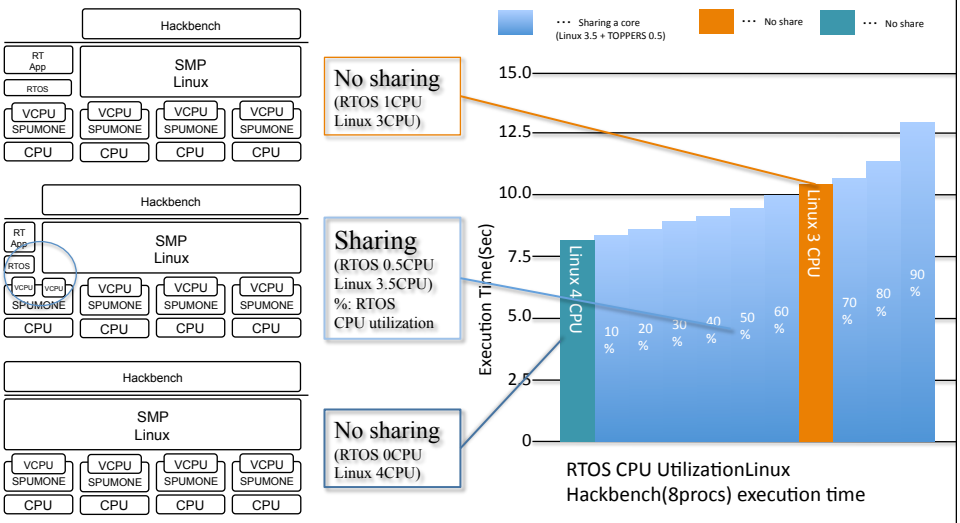


Dynamic Virtual CPU Management

- Let us assume that RTOS preempts VCPU2 while VCPU2 executes a critical section.
- VCPU1 may wait for entering the critical section with a spin lock until RTOS becomes idle and VCPU2 exits the critical section.
  - The situation is called “Lock Holder Preemption”.
- Existing approaches postpone the preemption until the critical section is exited.
  - This increases the jitters of RTOS.
- Cross-call executes a function simultaneously in all cores.

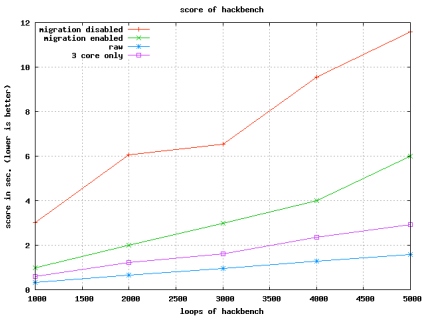


The Effect of Lock Holder Preemption



Current Solution and Problem

- Handling Multicore Anomalies
  - Avoiding Lock Holder Preemption
    - When an RTOS becomes runnable, but the preempted OS runs in a critical section, the RTOS is migrated to another core.
  - Handling Cross-Calls
    - We are considering to use priority boost while executing cross-calls and migrating VCPUs for Linux.
- Handling Dead Core
  - What happens if a core is dead while executing a critical section.

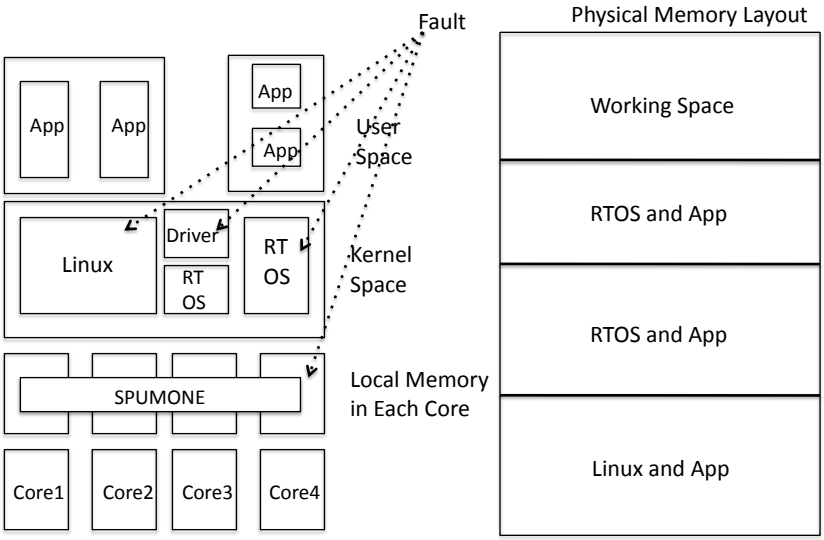


Current Open Questions

- Global scheduling policy
  - Mapping policy between PCPUs and VCPUs and migration policy of VCPUs.
    - Performance, Real-Time, Energy Consumption
  - Cache affinity of VCPU migration with a shared cache.
- Power management interface
  - Coordinating respective PM policies in respective OSes.
- Local VCPU Scheduling in each PCPU.
  - Cyclic Executive, Fair Scheduling
- Guest OS kernel: User space or Kernel space ?
  - Existing debugging tools for embedded systems assume that guest OS kernels run in the kernel space.

Proactive Recovery Management

- How to recover composed multiple platforms ?



Proactive Recovery Strategy

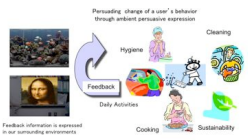
- Microrebooing: Various applications on RTOS do not have complex persistent states and it is possible to recover simply by rebooting.
  - Using proactive rebooting when something happens.
  - It is possible to put some small states in a shared persistent space to speed up the rebooting.
- Error Virtualization: In a case of Linux, rebooting may be postponed until Linux becomes idle.
  - When an error occurs in the kernel, it is translated to an error of a system call or a signal to an application as much as possible.
  - By scheduling rebooting, it is possible to reduce power consumption or decrease the user's frustration.
    - It is optimistic recovery strategy, and the approach does not always work well.

Building Reliable Application

Intelligent Robot

Sensor Network

Digital TV

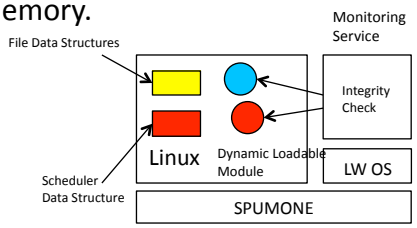


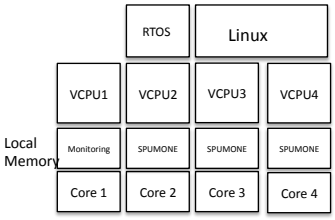
Persuasion Platform

- Intelligent Robot
  - If the rebooting time is sufficiently short, rebooting can recover the subsystem.
  - If not, a system should be decomposed into small subsystems, and each subsystem can be rebooted if it loses the integrity.
- Digital TV
  - Need to consider user satisfaction.
  - Error virtualization.
- Sensor Network
  - Rebooting can recover the subsystem.
- Persuasion Framework
  - Rebooting can recover the subsystem.
- We are planning to investigate how our approach affects the entire reliability via fault injection.

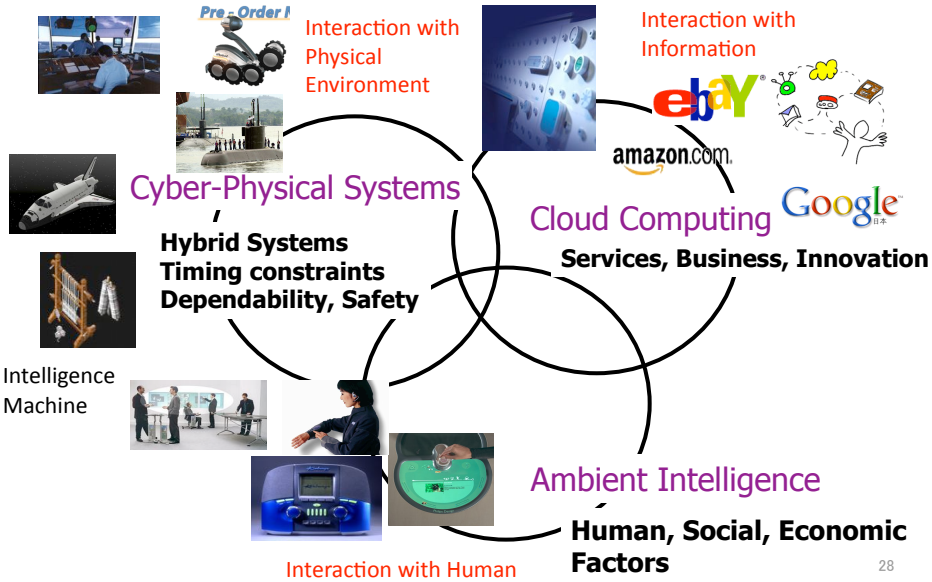
Integrity Management Service

- The integrity management service detects the violation of the integrity of the kernel and tries to repair the integrity autonomously.
- The integrity is violated due to a guest OS's internal errors, or due to other OSes' errors if there is no isolation among OS kernels.
- The integrity repair makes the kernel in a consistent state optimistically. The complete integrity can be ensured by a rejuvenation strategy.
- The integrity management services is isolated by using a location memory.





Towards Next Step Composability



Thank you!

Tatsuo Nakajjima
tatsuo@dcl.info.waseda.ac.jp