Introducing the Civil Infrastructure Platform

Yoshitake Kobayashi and Urs Gleim

Embedded Linux Conference, San Diego, April 5, 2016
Civil Infrastructure Systems are technical systems responsible for supervision, control, and management of infrastructure supporting human activities, including, for example,

- Electric power generation
- Energy distribution
- Oil and gas
- Water and wastewater
- Healthcare
- Communications
- Transportation
- Collections of buildings that make up urban & rural communities.

These networks deliver essential services, provide shelter, and support social interactions and economic development. They are society's lifelines.  

1) adapted from https://www.ce.udel.edu/current/graduate_program/civil.html
# The evolution of civil infrastructure systems

## Core characteristics

<table>
<thead>
<tr>
<th>Industrial gradeness</th>
<th>Maintenance costs</th>
<th>Technology changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>▪ Reliability</td>
<td>▪ Low maintenance costs for commonly used software components</td>
<td>▪ Proprietary nature</td>
</tr>
<tr>
<td>▪ Functional Safety</td>
<td>▪ Low commissioning and update costs</td>
<td>▪ Systems are built from the ground up for each product</td>
</tr>
<tr>
<td>▪ Security</td>
<td></td>
<td>▪ little re-use of existing software building blocks</td>
</tr>
<tr>
<td>▪ Real-time capabilities</td>
<td></td>
<td>▪ Closed systems</td>
</tr>
</tbody>
</table>

## Sustainability

| Product life-cycles of 10 – 60 years

## Conservative update strategy

| Firmware updates only if industrial gradeness is jeopardized |
| Minimize risk of regression |
| Keeping regression test and certification efforts low |

## Development costs

| Don’t re-invent the wheel |

## Development time

| Shorter development times for more complex systems |

## Business needs

## Technology changes

### Commodity

- Increased utilization of commodity (open source) components, e.g., operating system, virtualization
- Extensibility, e.g., for analytics

### Stand-alone systems

- Limited vulnerability
- Updates can only be applied with physical access to the systems
- High commissioning efforts

### Connected systems

- Interoperability due to advances in machine-to-machine connectivity
- Standardization of communication
- Plug and play based system designs

### Commoditization

- Increased utilization of commodity (open source) components, e.g., operating system, virtualization
- Extensibility, e.g., for analytics

### Technology changes

- Proprietary nature
- Systems are built from the ground up for each product
- Little re-use of existing software building blocks
- Closed systems

### Business needs

- Don’t re-invent the wheel

### Development time

- Shorter development times for more complex systems

### Core characteristics

- Industrial gradeness
- Sustainability
- Conservative update strategy
Things to be done

• Join forces for commodity components
  • Ensure industrial gradeness for the operating system platform focusing on reliability, security, and functional safety.
  • Increase upstream work in order to increase quality and to avoid maintenance of patches

• Share maintenance costs
  • Long-term availability and long-term support are crucial

• Innovate for future technology
  • Support industrial IoT architectures and state-of-the-art machine-to-machine connectivity
Comparison with existing Alliances

Other domains already benefit from collaborative development: drive instead of follow!

- Development speed for shorter product cycles
- High Software quality due to intense reviews and high test coverage (Linus’s law)
- Standard platforms enable ecosystems (e.g. for development tools, system extensions, new business models)

In many domains competing companies collaborate in alliances already. (GENIVI, for example)
Civil Infrastructure Platform to provide software building blocks that support reliable transportation, power, oil and gas, and healthcare infrastructure

Establish an open source “base layer” of industrial grade software to enable the use and implementation in infrastructure projects of software building blocks that meet the safety, reliability, security and maintainability requirements.

- Share development effort for development of industrial grade bases systems.
- Fill the gap between capabilities of the existing OSS and industrial requirements.
- Reference-implementation consisting of
  - Specification of on-device software stack and tools infrastructure
  - Linux kernel, file system, etc. selected reference hardware
  - Build environment and tools for companies to build their own distribution.
  - Test framework and test cases
  - SDK and APIs
- Trigger development of an emerging ecosystem including tools and domain specific extensions.

Initial focus will be on establishing a long term maintenance infrastructure for selected Open Source components, funded by participating membership fees.
Scope of activities

User space

- App container infrastructure (mid-term)
- App Framework (optionally, mid-term)

Middleware/Libraries

- Domain Specific communication (e.g. OPC UA)
- Shared config. & logging

- Safe & Secure Update
- Monitoring
- Security

- Real-time support
- Real-time / safe virtualization

Linux Kernel

- Tools
  - Build environment (e.g. yocto recipes)
  - Test automation
  - Tracing & reporting tools
  - Configuration management
  - Device management (update, download)
  - Application life-cycle management

- Concepts
  - Functional safety architecture/strategy, including compliance w/ standards (e.g., NERC CIP, IEC61508)
  - Long-term support Strategy: security patch management
  - Standardization collaborative effort with others
  - License clearing
  - Export Control Classification

Kernel space

On device software stack

Product development and maintenance
**Out of scope:**
- Enterprise IT and cloud system platforms.

Reference hardware for common software platform:
- Start from working the common HW platform (PC)
- Later extend it to smaller/low power devices.

1) Typical configurations Q1/2016

---

**Target Systems**

<table>
<thead>
<tr>
<th>Networked Node</th>
<th>Embedded Control Unit</th>
<th>Embedded Computer</th>
<th>Embedded Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>M0/M0+/M3/M4</td>
<td>M4/7,A9,R4/5/7</td>
<td>ARM A9/A35,R7</td>
<td>ARM A53/A72</td>
</tr>
<tr>
<td>Quark MCU</td>
<td>Quark SoC</td>
<td>Atom</td>
<td>Core, Xeon</td>
</tr>
<tr>
<td>8/16/32-bit, &lt; 100 MHz</td>
<td>32-bit, &lt;1 GHz</td>
<td>32/64-bit, &lt;2 GHz</td>
<td>64-bit, &gt;2 GHz</td>
</tr>
<tr>
<td>n MiB flash</td>
<td>n GiB flash</td>
<td>n GiB flash</td>
<td>n TiB flash/HDD</td>
</tr>
<tr>
<td>&lt; 1 MiB</td>
<td>&lt; 1 GiB</td>
<td>&lt; 4 GiB</td>
<td>&gt; 4 GiB</td>
</tr>
<tr>
<td>Arduino class board</td>
<td>Raspberry Pi class board</td>
<td>SoC-FPGA, e.g.Zync</td>
<td>industrial PC</td>
</tr>
<tr>
<td>Sensor, field device</td>
<td>control systems</td>
<td>special purpose &amp; server based controllers</td>
<td></td>
</tr>
<tr>
<td>PLC</td>
<td>gateways</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>multi-purpose controllers</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

1) Typical configurations Q1/2016

... 4 Device class no.
Relationship between CIP and other projects

Civil Infrastructure Platform

Member companies

Developers

- CIP FTE’s
- Developers from member companies

Budget

CIP source code repositories

Optional: funding of selected projects

contribution

Collaborative Projects (e.g. RTL, Yocto, CII)

Existing project / distro

Existing project

New CIP sub-project

CIP Super Long Term Support Project

Open source projects (Upstream work)

Existing projects (unchanged)

Open source projects

CIP will do not only development for CIP but also fund or contribute to related upstream projects

- Import source code from open source project or existing distribution to CIP
- Backport patches from upstream to CIP version
Upstream first policy for implementation of new features

All delta from mainline should be treated as technical debt.

• No parallel source trees, directly discuss features in upstream projects.
• Upstream first implementation. Take this to declared stable.
• Then back-port to long-term support versions drive by CIP employee or CIP members.
Super Long Term Support - Motivation

**Upstream Kernel tree**
- Long-term support (LTS) (LTS): Backports bug fixes for 2 years
  - About 3 months
  - Approx. 2-5 years

**Long-term support initiative (LTSI)**
- Add extra functionality on LTS for embedded systems and support it for 2 years
  - Approx. 2-5 years
  - 10 years – 15 years

**Every company, every project**
- Backport of bug fixes and hardware support: the same work is done multiple times for different versions.

Release / Maintenance release
CIP kernel super long term support (SLTS) overview

**Kernel.org**
- **Upstream Kernel tree**
  - Approx. 3 months
- **Long-term support (LTS)**
  - Backports bug fixes for 2 years
  - Approx. 2-5 years

**CEWG**
- **Long-term support Initiative (LTSI)**
  - Add extra functionality on LTS for embedded systems and support it for 2 years
  - Approx. 2-5 years

**CIP**
- **CIP super long-term supported kernel**
  - Need to be maintained more than 10 years
  - Approx. every 3 years
  - Goal: 10 years – 15 years

- **Release / Maintenance release**

- **Backports, e.g. for SoC support reviewed by CIP**

- **After 5 years merge window for new features will be closed, CIP kernel changes focus to security fixes.**
Package categorization

- **CIP development packages**
  - The "development packages" provide a reproducible environment for building the CIP kernel and related packages
  - This category should include all build dependencies, debug tools and test tools for CIP kernel and CIP core components
  - This category might not require to have security fixes

- **CIP core packages**
  - CIP will provide super long-term support for this category

- **CIP Linux Kernel**
  - Linux kernel itself which CIP will maintain
  - CIP will provide super long-term support for this category

**Hardware (Development board / QEMU)**
Candidates for Super Long-term Maintenance

An Example minimal set of “CIP kernel” and “CIP core” packages for initial scope

<table>
<thead>
<tr>
<th>Core Packages (SLTS)</th>
<th>Super Long-term support</th>
<th>Maintain for Reproducible build</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Kernel (SLTS)</strong></td>
<td>• Kernel</td>
<td>• Git</td>
</tr>
<tr>
<td></td>
<td>• Linux kernel (cooperation with LTSl)</td>
<td>• pax-utils</td>
</tr>
<tr>
<td></td>
<td>• PREEMPT_RT patch</td>
<td>• Pciutils</td>
</tr>
<tr>
<td></td>
<td>• Bootloader</td>
<td>• Perl</td>
</tr>
<tr>
<td></td>
<td>• U-boot</td>
<td>• pkg-config</td>
</tr>
<tr>
<td></td>
<td>• Shells / Utilities</td>
<td>• Popt</td>
</tr>
<tr>
<td></td>
<td>• Busybox</td>
<td>• Procs</td>
</tr>
<tr>
<td></td>
<td>• Base libraries</td>
<td>• Quilt</td>
</tr>
<tr>
<td></td>
<td>• Glibc</td>
<td>• Readline</td>
</tr>
<tr>
<td></td>
<td>• Tool Chain</td>
<td>• sysfsutils</td>
</tr>
<tr>
<td></td>
<td>• Binutils</td>
<td>• Tar</td>
</tr>
<tr>
<td></td>
<td>• GCC</td>
<td>• Unifdef</td>
</tr>
<tr>
<td></td>
<td>• Security</td>
<td>• Zlib</td>
</tr>
<tr>
<td></td>
<td>• Opensssl</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Openssh</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:** The maintenance effort varies considerably for different packages.
CIP will increase development effort to create industrial grade commin base-layer

**Development plan**

- **Phase 1:**
  - Define supported kernel subsystems, arch.
  - Initial SLTS component selection
  - Select SLTS versions
  - Set-up maintenance infrastructure (build, test)

- **Phase 2:**
  - Patch collection, stabilization, backport of patches for CIP kernel packages
  - Support more subsystems
  - Additional core packages

- **Phase 3:**
  - Domain specific enhancements, e.g. communication protocols, industrial IoT middleware
  - Optionally: more subsystems
  - Optionally: more core packages
Milestones

• 2015:
  • Set up collaborative software project.
  • Proposals for base components collected and evaluated.
  • Maintenance strategy defined

• 2016:
  • Project launch announcement at Embedded Linux Conference 2016
  • Requirements defined, base use cases defined, technical & non-technical processes established (license clearing, long-term support), maintenance plan
  • Common software stack defined, related core projects agreed (e.g. PREEMT_RT, Xenomai), maintenance infrastructure set up
  • Domain specific extensions defined, tool chain defined, test strategy defined
  • Maintenance operational and running

• 2017:
  • Realization phase of selected components

• 2018:
  • Advancement, improvements, new features
Please join!

Maintainers wanted

Meet us at ELC

Yoshitake Kobayashi (Toshiba)
Jan Kiszka (Siemens)
Urs Gleim (Siemens)
Wolfgang Mauerer (Siemens)
Takuo Koguchi (Hitachi)
Paul Sherwood (Codethink)

Platinum Members

Silver Members

HITACHI
SIEMENS
TOSHIBA

CIVIL INFRASTRUCTURE PLATFORM
Civil Infrastructure Platform: Executive Summary

- Civil infrastructure systems are currently built from the ground up, with little re-use of existing software building blocks. However, existing software platforms are not yet industrial grade (in addressing safety, reliability, security and other requirements for infrastructure). At the same time, rapid advances in machine-to-machine connectivity are driving change in industrial system architectures.

- The Linux Foundation proposes the creation of the Civil Infrastructure Platform (“CIP”) as a Linux Foundation Collaborative Project. The Civil Infrastructure Platform will establish an open source “base layer” of industrial grade software to enable the use and implementation in infrastructure projects of software building blocks that meet the safety, reliability, security and other requirements of industrial and civil infrastructure.

- Initial focus will be on establishing a long term maintenance infrastructure for selected Open Source components, funded by participating membership fees.

- Mid-term focus will be extended to filling gaps commonly agreed addressing civil infrastructure systems’ requirements.

- The Civil Infrastructure Platform shall be hosted by the Linux Foundation as an internal Linux Foundation project, leveraging the resources and infrastructure of the Linux Foundation, including the Linux Foundation’s relationships with other open source projects.
Contact Information and Resources

To get the latest information, please contact:

• Noriaki Fukuyasu  fukuyasu@linuxfoundation.org
• Urs Gleim  urs.gleim@siemens.com
• Yoshitake Kobayashi  yoshitake.kobayashi@toshiba.co.jp

Other resources

• CIP Web site  https://www.cip-project.org
Questions?
Thank you!
Backup: Topics and related projects (subject to change)

Application support
- App Framework
- HMI Framework
- FW update
- App deployment

Middleware / Tools
- Coherent Security Mechanisms
- Domain specific communication
  - ZigBee
  - Avnu
  - Echonet
- Industrial special-purpose protocols
- IoT communication stacks
  - AllJoyn
  - IoTivity
  - OM2M
- Configuration/Device management
  - Configuration
  - Industrial Zeroconf

Linux Kernel
- Security
  - Anomaly detection
  - LSM
  - SELinux
- Functional Safety
  - Monitoring/error detection
  - SIL2LinuxMP
  - SIL3 support
- Userland Isolation
  - LXC
  - Cgroups

Safety
- Kernel Isolation
  - SafeG
  - Jailhouse
- Communication

Real-time support
- GPGPU/FPGA real-time
- Xenomai
- RT/non-RT communication
- PREEMPT-RT

Monitoring / Tracing
- Ftrace
- ktap
- RAS
- Monitoring / Tracing

Heterogeneous Computing
- SoC FPGA

Configuration/Device management
- Configuration
- Industrial Zeroconf

Security
- Monitoring/error detection
- SIL2LinuxMP
- SIL3 support

Userland Isolation
- LXC
- Cgroups

Real-time support
- GPGPU/FPGA real-time
- Xenomai
- RT/non-RT communication
- PREEMPT-RT

Hardware / SoC (x86 or ARM based)

Legend
- To be specified / implemented by CIP
- Integration / cooperation

Infrastructure and Services
- Build and production
  - Yocto Project

Testing
- LTP
- kselftest
- LTSI test
- CIP test suite

Support
- SLTS
- Backwards compatibility

Development process
- SIL2 support
- SIL3 support

Legal topics
- SPDX
- FOSSology

License Clearing
- Export Control

Infrastructure and Services
- Build and production
  - Yocto Project

Testing
- LTP
- kselftest
- LTSI test
- CIP test suite

Support
- SLTS
- Backwards compatibility

Development process
- SIL2 support
- SIL3 support

Legal topics
- SPDX
- FOSSology

License Clearing
- Export Control

Legend
- To be specified / implemented by CIP
- Integration / cooperation