Real-Time and Security Requirements in IoT Operating Systems

Echtzeit 2016, Boppard, Germany

Maja Malenko and Marcel Baunach
malenko, baunach}@tugraz.at

17 November 2016

Institute for Technical Informatics
Embedded Automotive Systems Group
Graz University of Technology

Overview

- Dependable IoT
- Requirements for the IoT RTOSs
- Design choices for the IoT RTOSs
- Overview of existing OSs for the IoT
- Comparison of existing OSs for the IoT
- What is missing concerning IoT?
- Conclusion
The Internet of Things
… will be everywhere!

- 50B+ connected devices by 2020
- Is one out of the “Ten most disruptive Technologies”
- Will impact the most critical areas of our daily life:
  - City management
  - Power grids & plants
  - Water management
  - Healthcare
  - Transportation (road, rail, air, water, space)
  - Infrastructure management, etc.

⇒ The IoT must not fail ⇐

We need to provide guarantees on dependability:

- **Security** (integrity, confidentiality)
  → no misuse or alteration
- **Safety**
  → no catastrophic consequences
- **Real-Time** (availability, reliability)
  → continuous correct, accurate and timely service
- **Maintainability**
  → flexible to adaptations and updates

⇒ Cooperation across scientific disciplines
to provide rigorous methods on designing, implementing and operating the IoT.
“Dependable Internet of Things”

Research Project at TU Graz since 2016

- SP1: Dependable Wireless
  Guaranteed communication and localization performance (6P)

- SP2: Dependable Computing
  Guaranteed timely and secure software execution (4P)

- SP3: Dependable Composition
  Guaranteed correctness and performance of the composed system (6P)

- SP4: Dependable Networked Control
  Guaranteed distributed control performance (4P)

10 Key Researchers across EE/CS Faculties + 10 PhD

http://www.tugraz.at/projekte/dependablethings/home

Subproject 2: Dependable Computing

**Challenge:** Computing Platform for mixed Real-time and Security

- Operating Systems for dynamic composition of embedded real-time software
- Processor Extensions for secure software execution and data/flow integrity
- Compositional RTOS Kernel
- Flexible MCU Architecture

http://www.tugraz.at/projekte/dependablethings/home
Requirements for the IoT RTOSs

- What makes a good IoT RTOS?
  - Dependable
    - Real-time
    - Security
    - Safety
  - Maintainability
    - Modular
    - Portable
  - Efficient
    - Resources
    - Usability
    - Connected

- State of the art
  - RTOSs often neglect security
  - RTOSs often neglect dynamical composition

Design choices for the IoT RTOSs

- Kernel architecture
  - Monolithic kernel
  - Modular kernel
  - Microkernel
  - Exokernel
- Programming model
  - Event-driven
  - Multithreading
- Scheduling model
- Memory management
- Portability
- Security
TinyOS

- Monolithic kernel
- Programming model
  - Event-driven
  - Implemented in nesC
- Two-level scheduling
  - RTC tasks
  - Event handlers
- Static memory allocation
- Optional
  - Preemptive TOSThreads
- Support for several network stacks

Contiki

- Modular kernel
- Programming model
  - Event-driven
- Two-level scheduling
  - Event handlers
  - Interrupts
- Dynamic memory management
- Optional
  - Preemptive multithreading
- Support for several network stacks
RIOT

- Microkernel
- Programming model
- Multithreading
- Scheduler
  - Preemptive, fixed priority
  - Tickless scheduler
- Memory management
  - Static memory allocation in the kernel
  - Dynamic memory allocation for applications
- Support for different communication stacks

FreeRTOS

- Minimalistic microkernel
- Programming model
  - Multithreading (multitasking)
- Scheduler
  - Preemptive, and cooperative
  - Fixed-priority tasks
- Memory management
  - Allocate memory once for all
  - On demand allocate and free memory
    - Constant sized
    - Variable sized
- No own network stack
- SafeRTOS
Zephyr

- Minimal hybrid micro/nanokernel
- Programming model
  - Cooperative fibers in nanokernel
  - Multithreading (multitasking) in microkernel
- Scheduler
  - Cooperative, priority-based fibers in nanokernel
  - Preemptive, priority-based tasks in microkernel
- Memory management
  - Microkernel supports dynamic memory allocation
  - Support in mind for many networking protocols

Sel4

- Microkernel
- The first-ever OS kernel that has been formally verified for
  - Functional correctness
  - Data confidentiality
  - Data integrity
  - Temporal integrity
- Sel4 concepts:
  - Capabilities
  - Kernel services (threads, IPC, capability spaces, interrupts)
  - Kernel objects (CNodes, TCBs, IPC endpoints, VA objects, interrupt objects, untyped memory)
    - Fixed size after creation
    - Untyped memory object used to create new typed objects
  - Preemptive, RR scheduler
### Commercial OSs

- Real-time
- Safety and security features
- POSIX-compliant

- QNX Neutrino
  - Microkernel
  - Built-in security features
- VxWorks
  - Monolithic kernel
  - Certified for real-time, reliability and security
- LynxOS
  - Monolithic kernel
  - LynxOS-178 – safety-critical
  - LynxSecure – security-critical

### Comparison

- Designed with low-level constrained embedded devices in mind
- Developed in C
  - Except TinyOS
- Mostly microkernel design
  - Less prone to errors, modular, easier verification
- Real-time
  - Except TinyOS and Contiki
  - Event-based programming
- Support multiple network protocols and multiple processor architectures
- No memory protection
  - Except seL4
SmartOS

A multi-tasking & multi-core operating system for compositional embedded applications:

- Knowledge of software design
- Handling of synchronous/periodic events
- Compliance with (hard/hot) timing constraints
- Dynamic composition
- Change MCU and application
- Exclusive resources
- System time
- IPC handling
- Exceptions
- Multi-core

Ports:
- Atmega128
- MSP430
- SuperH
- Aurix
- MosartMCU

Malenko, Baunach: "Real-Time and Security Requirements for the Internet of Things Operating Systems"
Baunach, Gomes, Mauroner: "Collaborative Resource Management for Multi-Core AUTOSAR OS"
Baunach: "Advanced Timestamping for pairwise Clock Drift Detection in Wireless Sensor/Actuator Networks"

Conclusion

- IoT imposes new requirements
- Existing RTOSs lack many IoT functionalities

- Challenges to be met
  - Dynamic composition of applications and services
    - Guarantee compatibility and interoperability
  - Dynamic Resource Sharing
  - Security risks

- An RTOS must evolve to deliver sustainability