Simplify Your Integration: Persistent Publish/Subscribe For Embedded Applications
Yi Zheng, Product Manager

2012 QNX KOREA TECHNOLOGY INNOVATION CONFERENCE – APRIL 18

2012 QNX 코리아 기술 혁신 컨퍼런스
2012년 4월 18일
Introduction: Challenges of Complex Systems

- Reliability & Availability
  - OS
- Design flexibility
- Scalability
- Connectivity
  - Network, Multiple devices and technologies
- HMI design
  - Integration of multiple technologies (OpenGL ES, Qt, etc.)
- Resource separation (e.g. sandbox)
- Branding and re-purposing core product
- Development time
- Certification
  - IEC 62304, IEC 61508, ISO 13485, ISO 1497, etc.
We’ll Learn About ...

• Messaging models
  – Asynchronous
  – Synchronous
  – Persistent Publish/Subscribe

• What is PPS?

• How PPS works

• How PPS can help with the challenges of designing and building complex systems
Asynchronous Messaging

• Well known and widely implemented
• Not ideal for systems with many devices and components
• Low-level solution that:
  – Pushes error handling, end-to-end semantics and buffer management up to the application level
  – Need protocols to ensure:
    • Correct behavior of messaging across all applications
    • Sufficient memory for applications
Synchronous: Send/Receive/Reply

• Very valuable in real-time environments
  – Many processes require responses to their messages before they proceed
  – System framework handles messaging errors and message buffers

• Not ideal for systems with many devices and components
  – Every server communicates directly with its clients and must know how to respond to all client messages
  – Change to one software component may require changes to others
  – Does not scale easily
Publish/Subscribe

• Publish/Subscribe

• 20+ years: K. P. Birman and T. A. Joseph (1987)
  – Virtual synchrony: “a fault-tolerant asynchronous bulletin board mechanism”

• Many examples of Publish/Subscribe implementations
  – Nortel Networks: network monitoring and reporting systems (DMS-100, etc.)

• ACM (Association for Computing Machinery) www.acm.org
  – Hundreds of papers

• a.k.a The Observer Pattern
Persistent Publish/Subscribe: PPS

- Publish/Subscribe with persistence across reboots
- Embedded applications that support
  - Many devices and software components
  - Sophisticated HMI combines different technologies:
    - Qt
    - OpenGL ES
    - C
PPS Case Study: Smart Energy Manager

• Design flexibility
• HMI design
  – HTML5
  – Crank Storyboard
  – Elektrobit Guide

• Connectivity
  – Zigbee wireless network
  – Insteon power-line based Home Area Network (HAN) for lighting control
  – IP video cameras
  – Internet
PPS Case Study: Medical Data Aggregator

- Design flexibility
- Scalability
- Connectivity
  - Database
  - BlackBerry PlayBook
- HMI design
  - Qt
  - C
  - Adobe Air

- Some components
  - Blood pressure
  - Spirometry
  - Pulse oximeter
  - ECG
Messaging Architecture

- Loosely-coupled publisher-subscriber architecture
- Publishing is asynchronous
- PPS client
  - Publish only
  - Subscribe only
  - Publish and subscribe
  - Subscribe to multiple objects
PPS Clients

• Must know which PPS objects are of interest
  – Publishers must know what to publish and when
  – Subscribers must know to which objects they must subscribe, and which object attributes are of interest
• Error and buffer management only for
  – `open()`, `read()` and `write()` POSIX API calls
  – Confirming that they can make sense of what they read
  – Setting reads to be blocking or non-blocking
• Need to know only that
  – Message has been read
  – Able to parse what was read
• Subscribers use `read()` calls to retrieve objects
  – Ergo: don’t need to manage buffers for these objects
PPS Objects

- Integrated into the PPS filesystem pathname space
  - Publishers modify objects and their attributes, and write them to the filesystem
  - When a publisher changes an object, the PPS service informs all clients subscribed to that object of the change
- Publishers can use the same object to communicate to all subscribers to that object
- Can have multiple publishers and subscribers
Binary or Human-readable Objects?

• PPS can use either binary or human-readable objects
• Human-readable objects and attributes
  – Benefits to development and debugging may outweigh the cost of the larger objects
• Debug from the command-line
  – Filesystem utilities
  – `cat` for subscribe
    ```bash
    cat /pps/media/PlayCurrent
    cat /pps/media/.all?wait
    ```
  – `echo` for publish
    ```bash
    echo "attr::value" >>/pps/objectfilename
    ```
• Simple script
  – Subscribes to an object
  – Prints out debugging information, including PPS object and attributes
Push or Pull Publishing?

- **Push** publishing system
  - Publishers push data to objects
  - Subscribers read data upon notification, or at their leisure
Push or Pull Publishing?

- **Pull** publishing system
  - Some data changes too quickly for push publishing
    - e.g. packet counts on an interface
  - Subscriber opens object with pull option
  - Issues `read()` call

- Publishers to the object notified to write current data
- Subscriber's read blocks, then returns with the new data
- On-demand publishing
Under the Hood: Smart Energy Management System
Under the Hood: Medical Data Aggregator
Persistence

- Needs reliable filesystem
  - Hard disk, NAND or NOR Flash, custom filesystem
- Maintains data across reboots
- Saves objects to persistent storage
  - On demand
  - At shutdown
- Restores objects on startup
  - Immediately
  - First access (deferred loading)
- Simplifies startups
Design Flexibility

• Publisher and subscriber do not know each other
  – Only connection is through an object
• Flexibility when designing a system
• Data Flow and Module Connection Points
  – Not hardcoded
  – Not directly linked
• Delay decisions on Module Connection Points and Data Flow until runtime
System Scalability

- Add components without increasing system complexity
- Only need to determine
  - What the new components publish
  - What data they need other PPS clients to publish
- No fine-tuning of APIs is required
Conclusion

- PPS messaging architecture
  - Flexible design
  - Scalability
  - Easily add devices and software components
  - Connectivity
  - Change, adapt, expand without touching the original
  - Design stability