IPMC Firmware

Fatih Bellachia, Thierry Bouedo, Sébastien Cap, Nicolas Dumont-Dayot, Sylvain Lafrasse, Nicolas Letendre, Thibault Guillemin, Alexis Vallier, Isabelle Wingerter

UNIVERSITÉ SAVOIE MONT BLANC

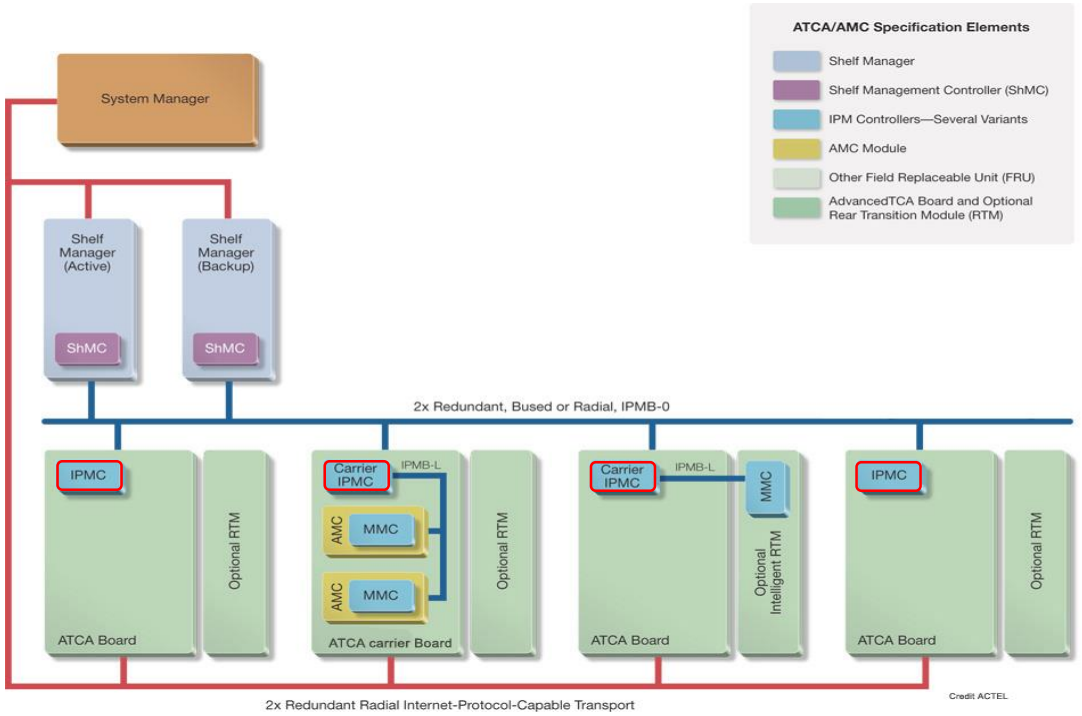
cnrs IN2P3 Les deux infinis

Software project name

INTELLIGENT PLATFORM MANAGEMENT CONTROLLER SOFTWARE

IPMC
Overview

The IPMC supports an *intelligent* hardware management system for ATCA board and ATCA carrier board (see Figure) which provides the ability to manage the power, cooling, and interconnect needs of intelligent devices; to monitor events; to log events to a central repository and also the ability to manage the mezzanine modules according to user's implementation, as well as the communication with the Shelf Manager.



IPMC

Specifications

The IPMC software solution is fully compliant with the following specifications:

- IPMI v1.5 *(document revision 1.1)* and some relevant subset of IPMI v2.0 *(document revision 1.0)*.
- PICMG 3.0 R3.0 *(AdvancedTCA™ base specification)*.
- AMC.0 R2.0 *(AdvancedMC™ base specification)*.

Software Environment

Features

- Linux host development
- 32-bit ARM Cortex-M4 microcontroller
- Written in standard ANSI C
- GCC (4.7.0) tool chain
- Open Source Configuration Management environment: - CMT
- FRU (ATCA board) generation utility (using M4 preprocessor)
- OpenOCD (0.9.0) utility (Linux/Windows)
  - Need USB to JTAG interface Debug-Adapter-Hardware
    - Olimex ARM-USB-TINY-H
    - NGX technology
    - …

Software design

Focus

- Distributed bare-metal application
- Event (message) driven architecture
- Design of the IPMC firmware is based on the components (modules)

Intelligent Platform Management Controller
Mezzanine

The IPMC supports an intelligent hardware management system for ATCA board and ATCA carrier board. The IPMC microcontroller control and monitor the operations and health of its host.

The IOIF microcontroller provides an extended I/O interface and deals with non-IPMC features.

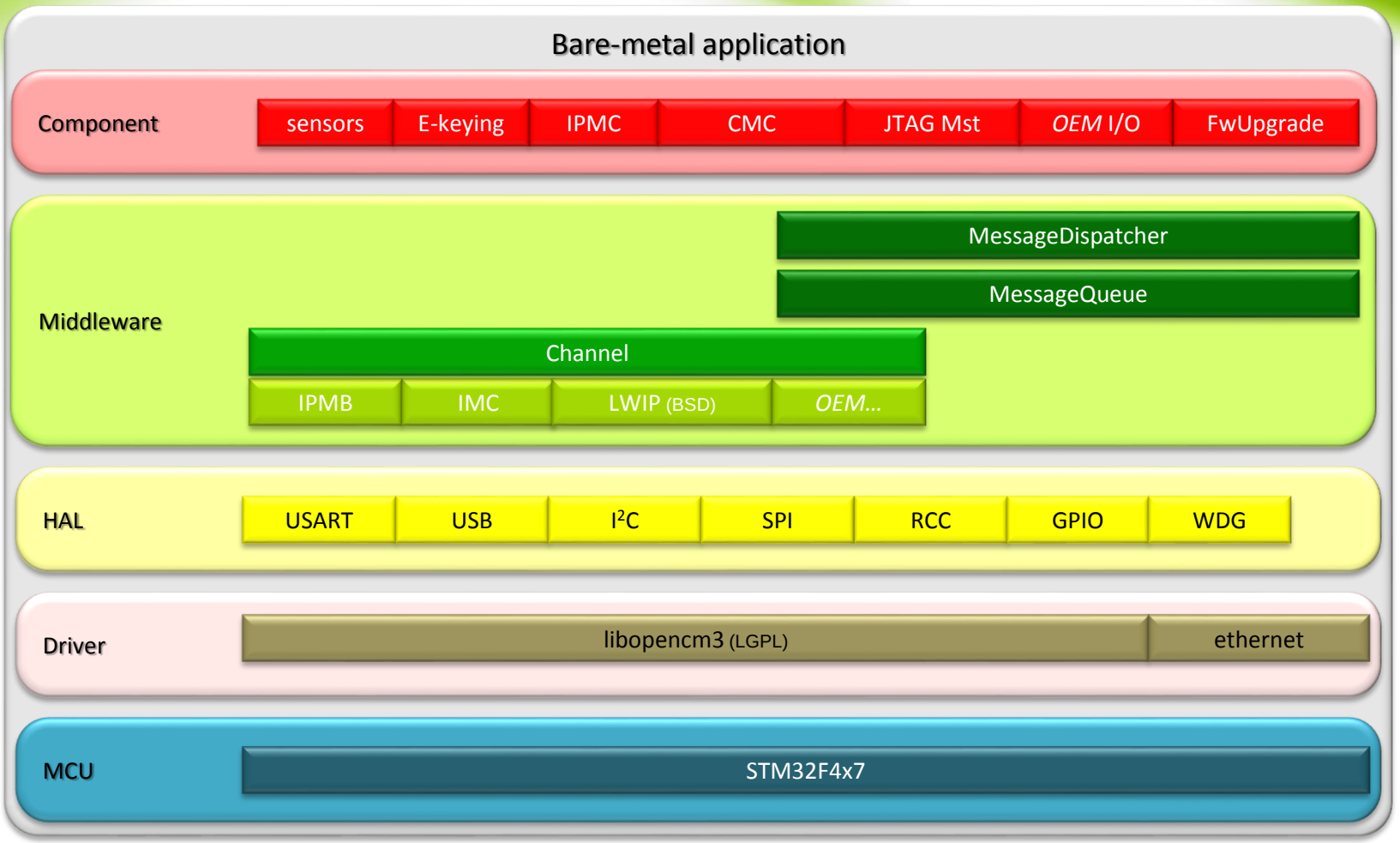
Modules

- Master Board Management Controller (PICMG® ATCA 3.0 R3.0)
- Carrier Management Controller (PICMG® AMC.0 R2.0)

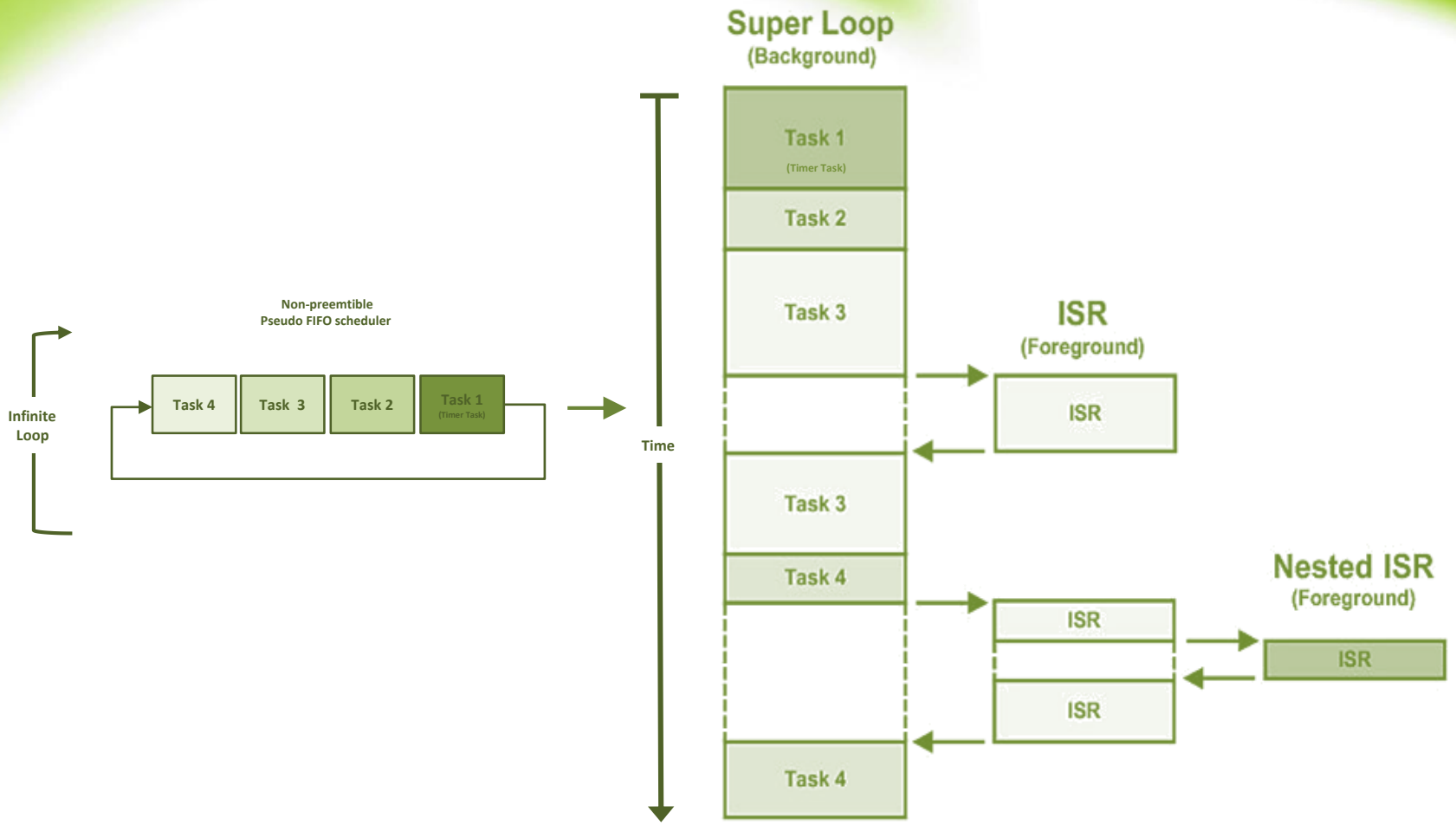
Modules

- Slave Board Management Controller
- Ekeying
- Sensors reading
- RTM Power Controller
- ATCA Power Monitor
- Command line Interpreter
- Firmware upgrade server
- Remote Command line interpreter
- Xilinx Virtual Cable server (JTAG master)
- TCP/UDP Echo server

Software architecture

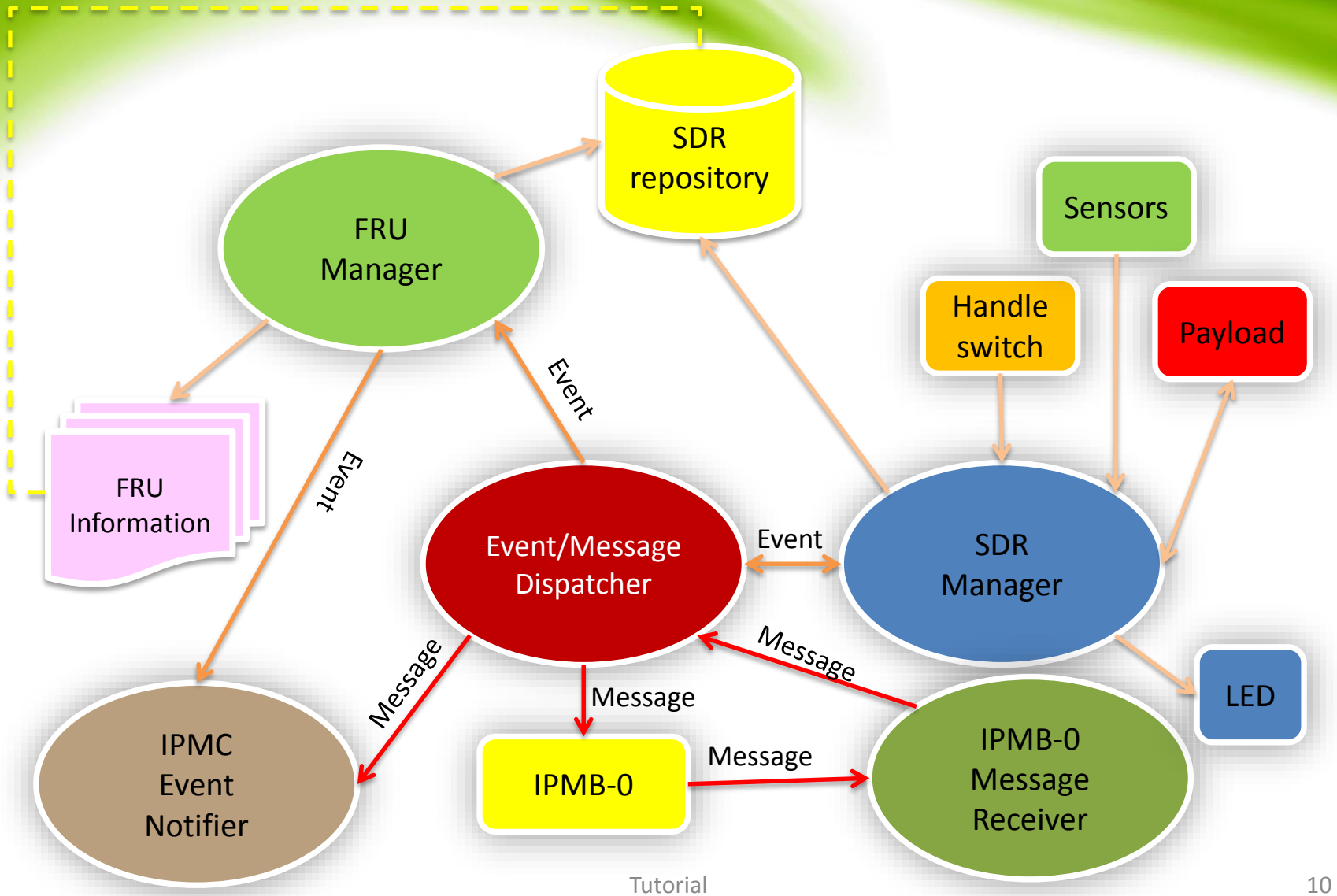


SuperLoop


Super Loop
(Background)
Task 1
(Timer Task)
Task 2
Task 3
Task 3
Task 4
Task 4
ISR
(Foreground)
ISR
Nested ISR
(Foreground)
ISR
ISR
ISR
Time
Non-preemtible
Pseudo FIFO scheduler
Infinite
Loop
Task 4
Task 3
Task 2
Task 1
(Timer Task)

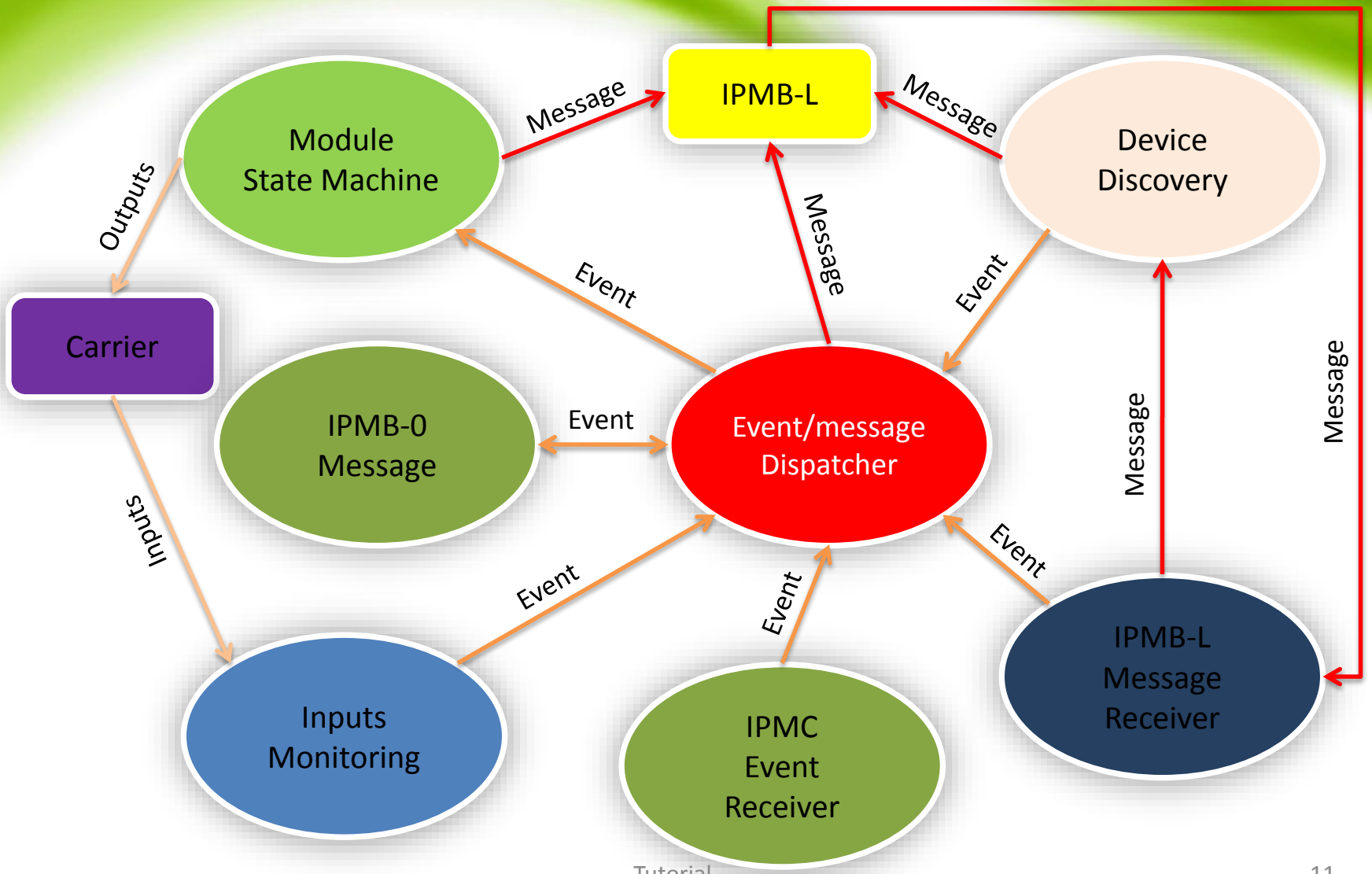
IPMC component

Intelligent Platform Management Controller synoptic

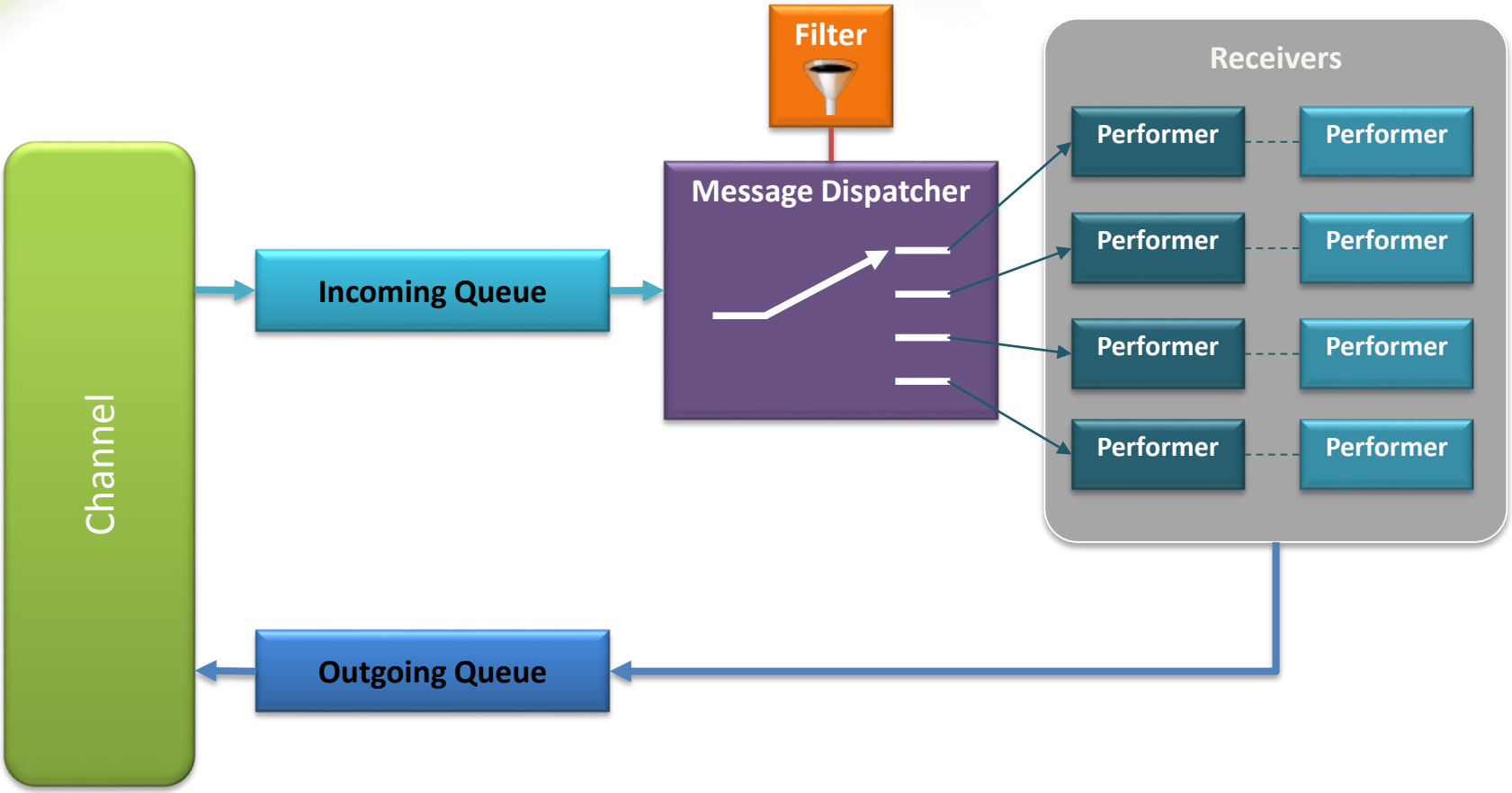


CMC component

Carrier Management Controller synoptic

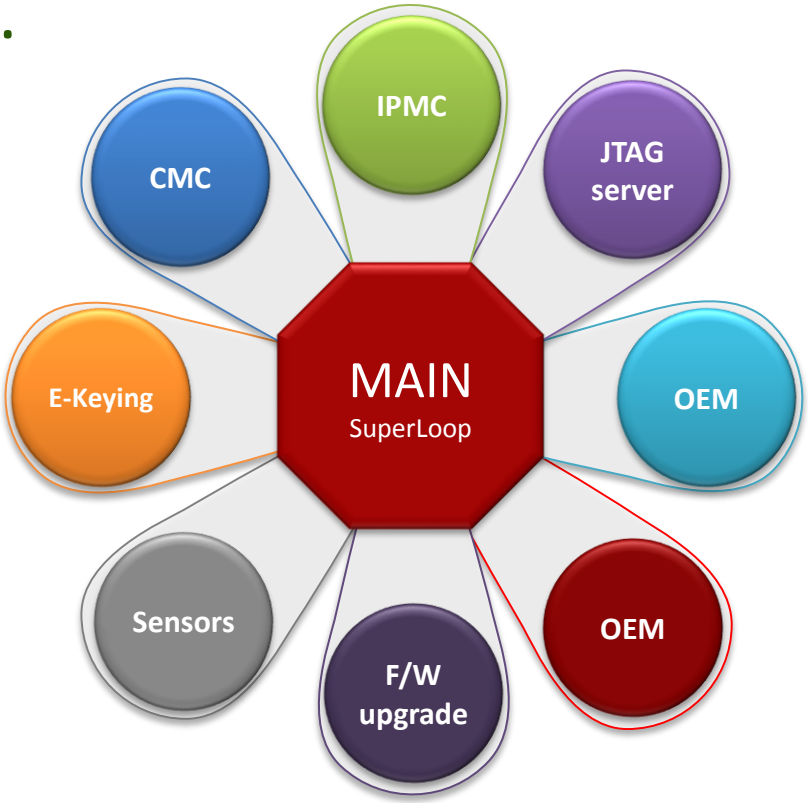


Message Dispatcher



Module

- The software "module" concept allows user to extend functionalities of IPMC without modifying existing code.



Module

- User need to provide 3 entry point functions and information.
  - Init
  - Cleanup
  - Process *(periodic or not)*
- E.g.

```
/*---------------------------------------------------------------------------*/
bool led0_init(void)
/*---------------------------------------------------------------------------*/
{
  LED_Off(LED_0);
  hTask = TimerTaskCreate(led0_blink, NULL, DELAY);

  return true;
}

/*---------------------------------------------------------------------------*/
void led0_process(void *pvArg)
/*---------------------------------------------------------------------------*/
{
  printf("\rLEDs [%c", bOn? '*': '\'');
  fflush(stdout);
}

MODULE_BEGIN_DECL(led0)
  MODULE_NAME("Blinky LED 0")
  MODULE_AUTHOR("fatih.bellachia@lapp.in2p3.fr")
  MODULE_INIT(led0_init)
  MODULE_CLEANUP(NULL)
  MODULE_PROCESS(led0_process, NULL)
MODULE_END_DECL
```

module manifest
see module/module.h

EEPROM

prom.c

- Used to store FRU/SDR information
- The Framework support EEPROM **M24256 or 24xx256**
- For an unsupported EEPROM the user must overwriting the following functions:
  - bool prom_init(void)
  - bool prom_reset(void)
  - int prom_erase(unsigned addr, int len)
  - bool prom_busy(void)
  - int prom_read(void *dst, unsigned src_addr, int len)
  - int prom_write(unsigned dst_addr, void *src, int len)
- By default FRU/SDR data are stored in 'IOIF' MCU memory.

Sensor

sensors.c

- The Framework support the following sensors
  - AD7414
  - LTC4151
  - LTC2499
  - IQ65033QMA10
- Register your sensor with the ResourceBroker library e.g.:

```
/*-------------------------------------------------------------------------*/
bool SensorsInit()
/*-------------------------------------------------------------------------*/
{
  RBResource_t stResource;

  // AD7414 Temperature
  stResource.ucChannelId  = CHANNEL_I2C_SENSOR;
  stResource.ucAddress    = SDR_AD7414_I2C_ADDR;
  stResource.ucIdentifier = SDR_NUM_AD7414_TEMP;
  stResource.fnInit       = InitSensorAD7414;
  stResource.fnRead       = ReadSensorAD7414;
  stResource.fnWrite      = WriteSensorAD7414;

  if (ResourceBrokerAddResource("AD7414 Temp", &stResource) == false)
    return false;

  …

  return true;
}
```

Channel #

I2C address

Unique sensor ID

User's functions

SDR

sdr_data.c

- Channel ID

  The channel IDs are defined in channel/channel.h

  CHANNEL_I2C_MGT (managed by IPMC)

  CHANNEL_I2C_SENSOR (managed by IPMC)

  CHANNEL_I2C_USER_IO

  CHANNEL_I2C_IPM_IO

  - <Full|Compact> Sensor Record

    set BYTE 7 [4:7] channel_num

- Sensor Number

  Unique number identifying the sensor *(e.g. see cfg_data.h)*

  - <Full|Compact> Sensor Record

    set BYTE 8 [0:7] sensor_number

- I2C address of sensor

  - Compact Sensor Record

    set BYTE 31 [0:7] OEM - Reserved for OEM use

  - Full Sensor Record

    set BYTE 47 [0:7] OEM - Reserved for OEM use

E-Keying

p2p_ekeying.c

- Register your backplane channel with ekeying library e.g.

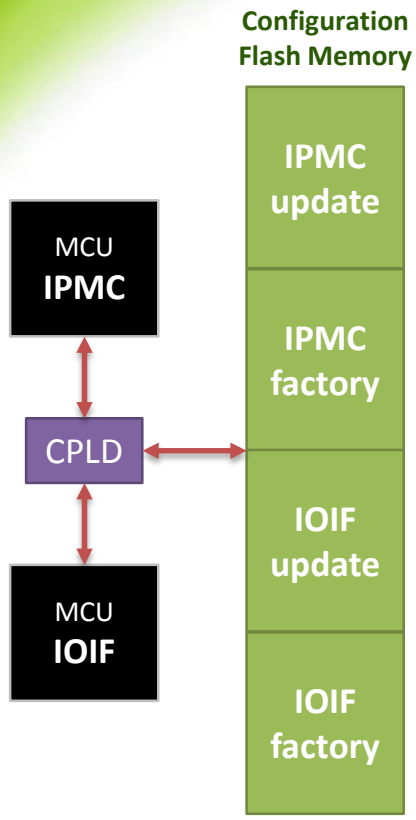
```
/*-------------------------------------------------------------------------*/
bool set_port_state_callback(linkDescriptor linkInfo, char state)
/*-------------------------------------------------------------------------*/
{
  if (linkInfo == 0x00001101)
    do something…
  else if (linkInfo == 0x00001102)
    do something else…

  …

  return true;
}

/*-------------------------------------------------------------------------*/
bool my_init(void)
/*-------------------------------------------------------------------------*/
{
  …
  EKRegisterFunc(0x00001101, set_port_state_callback);
  EKRegisterFunc(0x00001102, set_port_state_callback);

  …

  return true;
}
```

Base I/F – channel 1

Base I/F – channel 2

Firmware & Tools

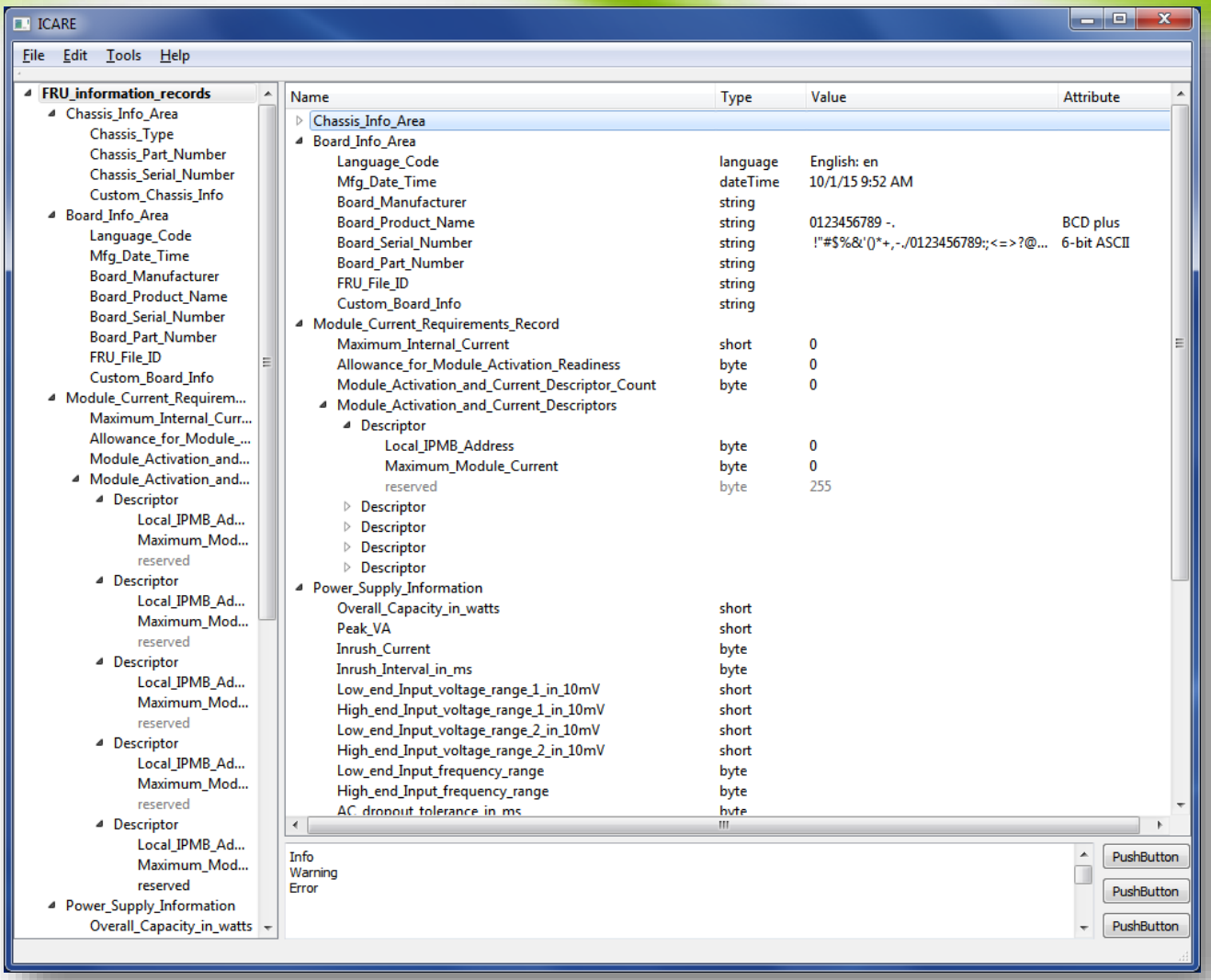
F/W Upgrade



- Factory Firmware will be initially stored in both the Configuration flash memory and micro-controllers internal flash memory.
- The factory firmware always remains available in the IPMC flash memory.
- The micro-controllers will revert to it after 3 failing consecutive attempts of upgrading the internal flash memory.
- Firmware update requires a new firmware in the IPMC CPLD.
  - Version 5.2 is available from the ICARE twiki page.
  - Requires the USB-JTAG Adaptor and openOCD tool for upgrading.
  - For those requiring it, IPMCs will be exchanged if you do not want to do the update yourselves.

Firmware & Tools

FRU/SDR generator

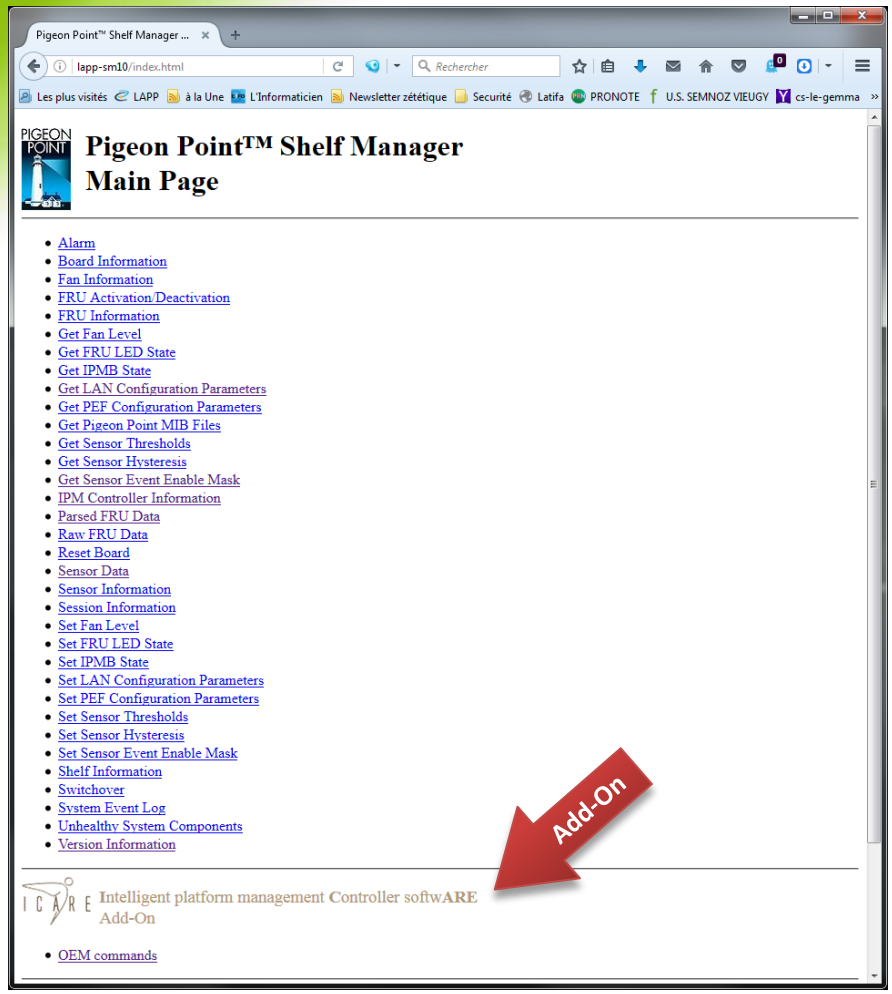


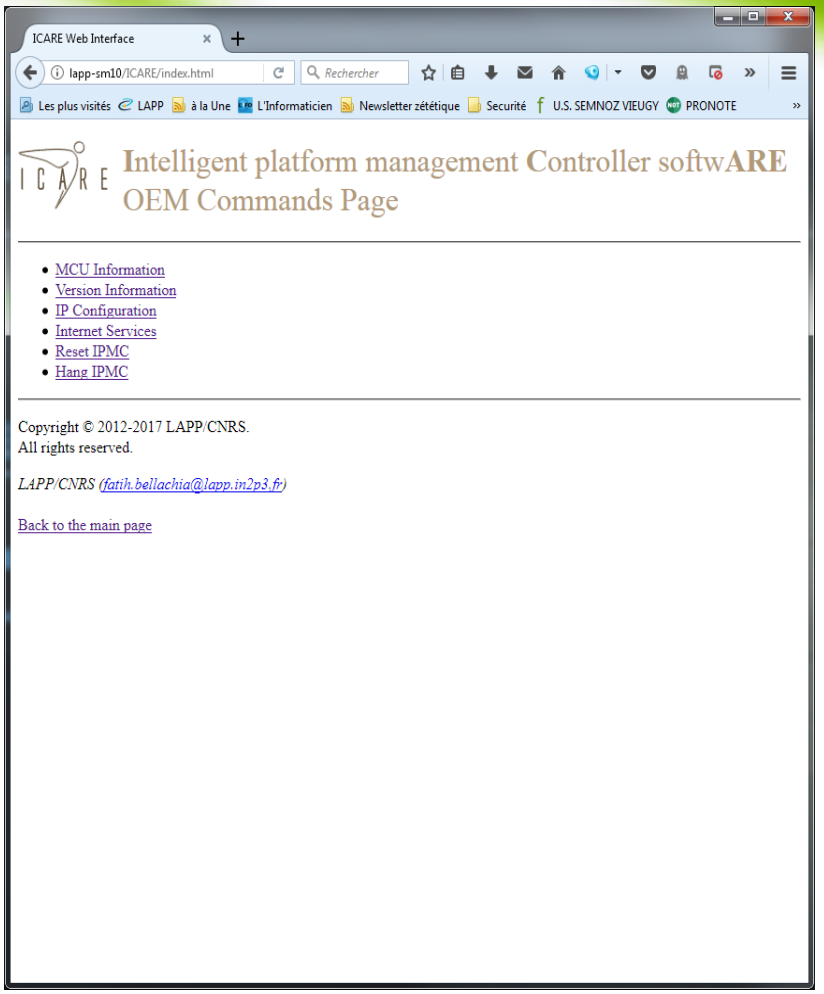
OEM Commands

- MCU Information
  - MCU ID
  - PCB version
  - Serial Number
- Version Information
  - Release version
  - Compiler version
  - Binary image name
  - Build version (i.e. Date/time)
  - List of used packages
    - Name + Version
- IP Configuration
  - MAC address
  - IP address
  - Link status
  - Duplex
  - Speed
- Internet services
  - List of services
    - Name + Protocol + Port + [Client: IP address + port]
- Reset IPMC
- Hang IPMC

Shelf Manager WEB Interface

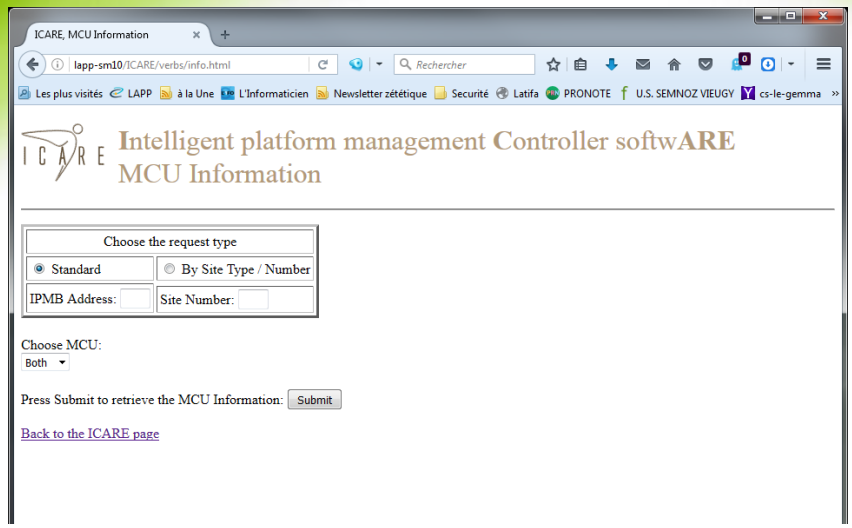
Upgrade

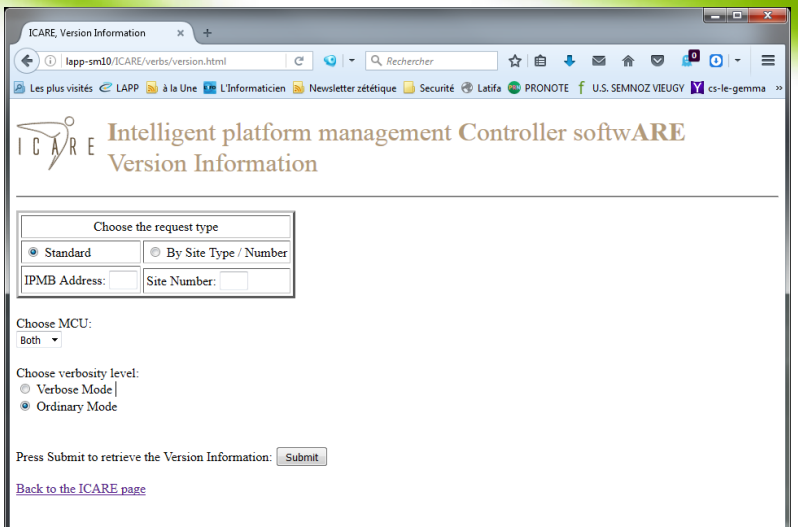


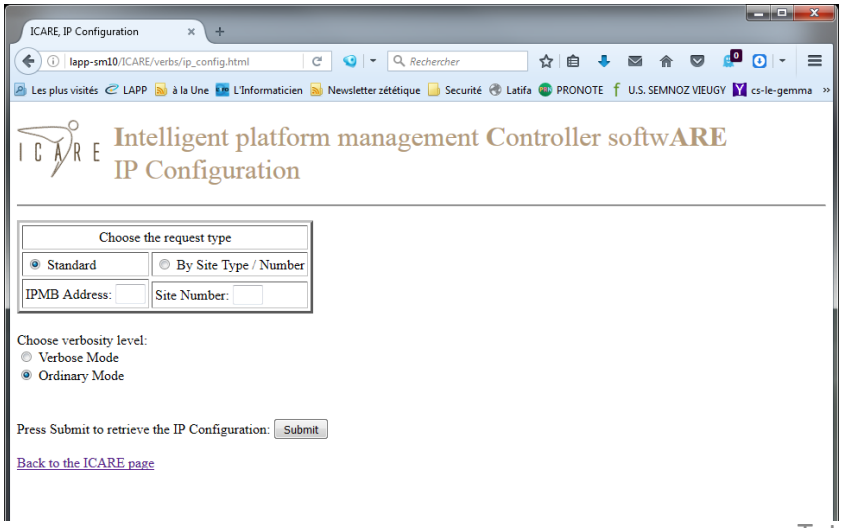


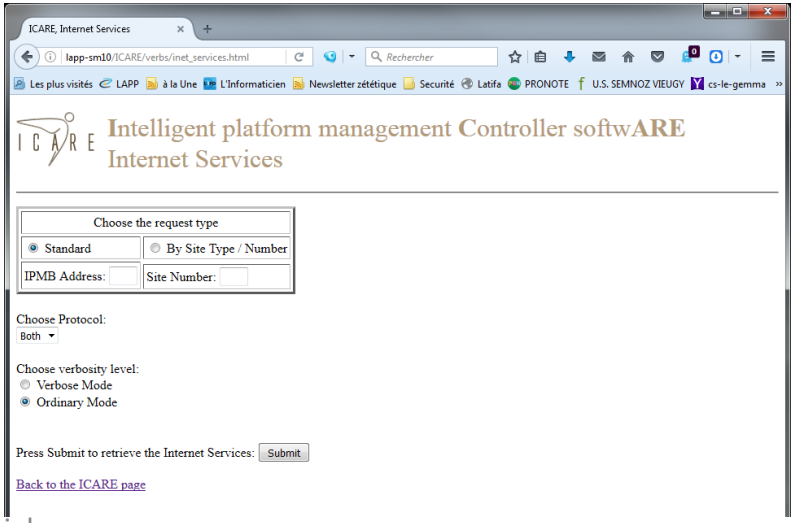
Shelf Manager WEB Interface

Upgrade









Software project area
ICARE-00-01-00