Embedding Real-time Object into Embedded Linux

2003-10-12

Moon Hae Kim, Konkuk University
Hyung-Seok Lee, ETRI

Contents

- Overview of National R&D Activities in Korea
- Linux based Object-oriented Real-time Operating Platform
  - R&D scope of KKU SWRC
  - TMO execution engines
    - TMO-Linux for server
    - TMOSM/Linux for client
- Introduction to ESTC projects
  - Digital Home Solutions
  - An Embedded Linux: Qplus/Esto
- Summary

Overview of National R&D Activities

- **Most of government branches try to build ubiquitous Korea (uKorea)**
  - **Good network infrastructure (high speed/mobile)**
  - **Strong manufacturing industries**
- **Development of embedded software technology becomes essential**
  - **MOST : 21C Frontier Research Program for Ubiquitous Computing and Network (2003.9 – 2012.8)**
    - Service scenario
    - Intelligent Object based Real-time Service Technique
    - Low-powered mobile computing platform
    - Ubiquitous network access
  - **MOCIE**
    - Support for SoC, Post PC, DTV, and Robot areas
    - Support for research institutes and university research centers
      - KETI (Korea Electronics Technology Institute)
        - » Ubiquitous Computing Research Center
        - » Center for Post-PC Industry Support
      - SNU ESRC (Embedded System Research Center)

- **MIC (Ministry of Information and Communication)**
  - Responsible for development of embedded software technology
  - Vision
    - The first country with a dream of “Embedded, Everywhere” fulfilled
    - One of the countries with the most advanced embedded software technology by the year 2007
  - Execution strategy
    - Move from catch-up strategy to leading strategy
      - » Catch-up stage: assembly and manufacturing
      - » Leading stage: original technology and international standards
    - Develop world-best products by adopting domestic embedded software for our competitive products, such as wired/wireless communication equipments and home appliances
    - Implement the “digital life” as early as possible by constructing a model *Digital Home* based on domestic embedded software

- SWOT analysis by MIC

S (Strength)	W (Weakness)
 - Good network infrastructure - Top-class technologies in consumer electronics - Agile adaptation to new technology	 - Brand image
O (Opportunity)	**T (Threat)**
 - Emerging markets (e.g. China) - Competent startup companies	 - Fierce competition - Small domestic markets

- Organizations for Embedded S/W
  - Digital Home Standardization Forum (4 divisions)
    - » Digital home system
    - » Media electronics
    - » Automation
    - » Sensor and health care
  - Embedded Linux Consortium (ELC) Korea
    - » Standardize the platform for embedded Linux
  - Wireless Internet Standardization Forum
    - » Establish the standard for domestic wireless Internet platform (WIPI)
  - Embedded software industry council
    - » Cooperate with industries for adapting standard embedded S/W technologies and developing embedded systems
- Collaboration with other government branches (Ministries)
  - Adapt the standard embedded S/W technologies to its own IT-related projects
- Utilization of Digital Media City (City of Seoul)
  - Used as the test environment in which embedded systems are built and new services are integrated

- Support for research institutes and university research centers
  - ETRI (Electronics and Telecommunication Research Institute) ESTC (Embedded S/W Technology Center)
    - » Technology Development Group for Embedded S/W Consists of 6 teams such as Embedded OS Research team
    - » Support team for Embedded S/W Industry
  - TTA (Telecommunications Technology Association)
    - » Standardization and Technology Certification
  - University ITRC (Information Technology Research Center) support program
    - » KKU SWRC (Software Research Center)
    - » SMU Center for Embedded S/W Development Environment
    - » ICU Sensor Network Research Center

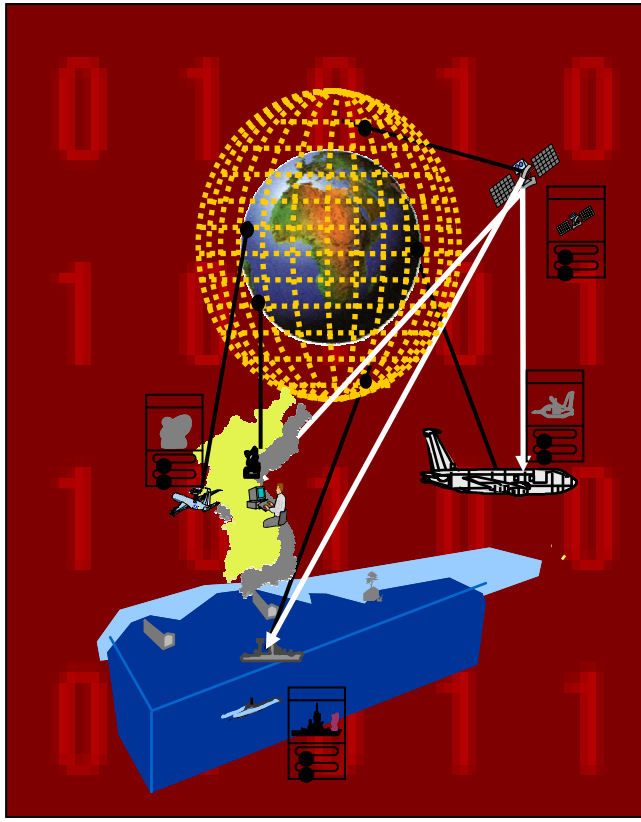
    etc.

Linux based Object-oriented Real-time Operating Platform

Software Research Center (SWRC)
Konkuk University (KKU)

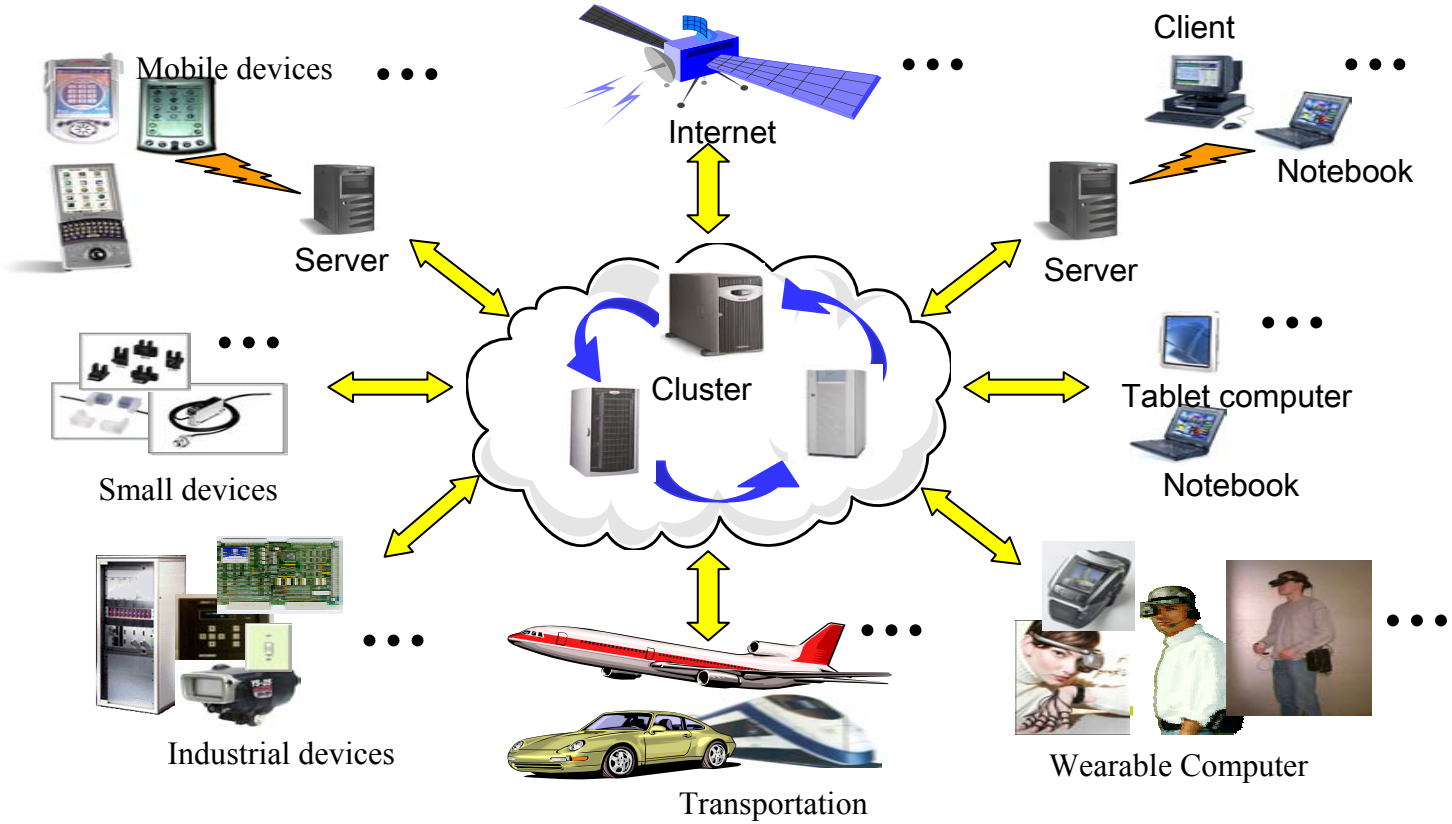
Vision

- Develop techniques and tools for ultra-reliable real-time distributed computing in the area of embedded (ubiquitous) applications.
  - TMO (Time-triggered Message-triggered Object) based approach
  - Towards Open Source
  - Reliable real-time OS and middleware
  - Multimedia middleware
  - Timing analysis, reliability analysis, modeling, and run-time monitoring tools
  - Real-time simulation technique and environment
- Collaborators
  - UCI DREAM Lab
  - ETRI Embedded S/W Tech. Center



R&D Scope

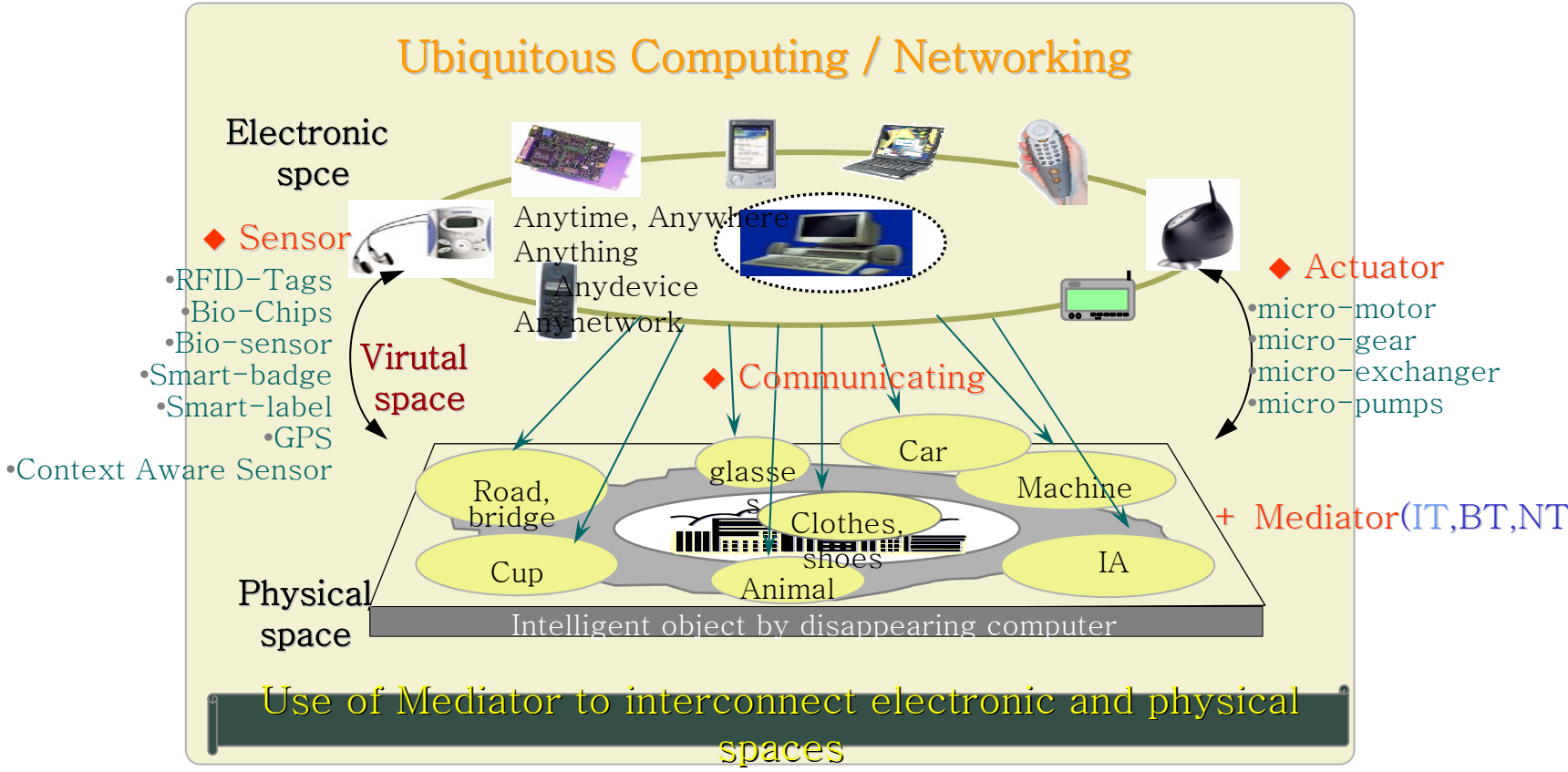
- Next-Generation Computing Platform
  - High-level programming for distributed real-time embedded systems
  - High dependability and ultra-precision real-time support



R&D Scope (cont'd)

▼ Next-Generation Computing Platform

- Kernel-level support for mobility and security (including protection)
- Seamless interconnection

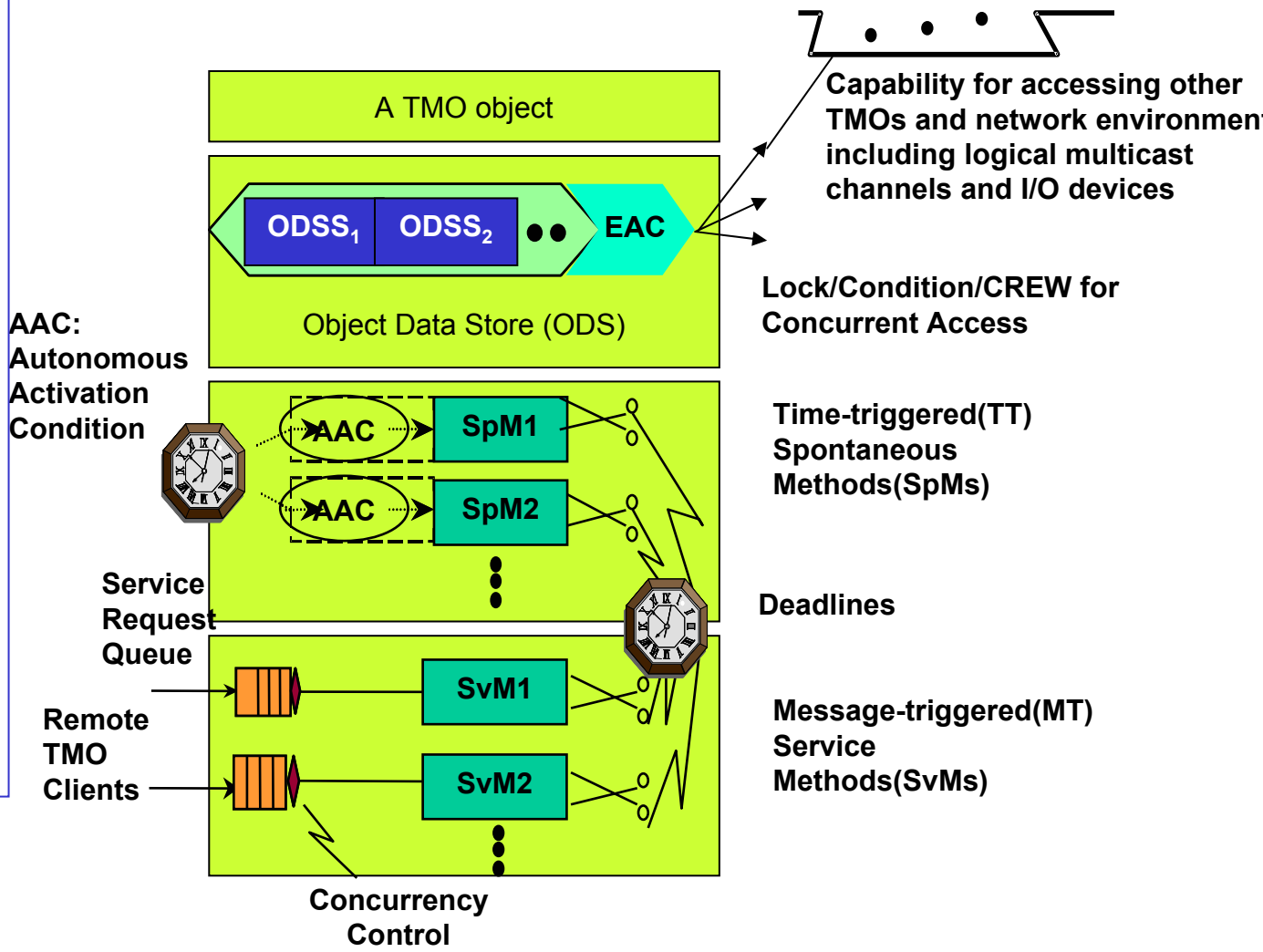


Current Research

- **Linux based Highly Reliable Distributed Real-time Operating Platform and Its Application Development Technology**
- **Focusing on the development of**
  - **Embedded computing platform technology**
    - Timeliness support based on the TMO (Time-triggered Message-triggered Object) model
    - Development of an execution environment for TMO structured S/W
    - Provision of an integrated development environment including timing analysis and monitoring capabilities (2003-2007)
    - Efficient resource management and multimedia communication support
  - **Reliable mobile network technologies such as mobile IPv4/IPv6 and ad hoc network**

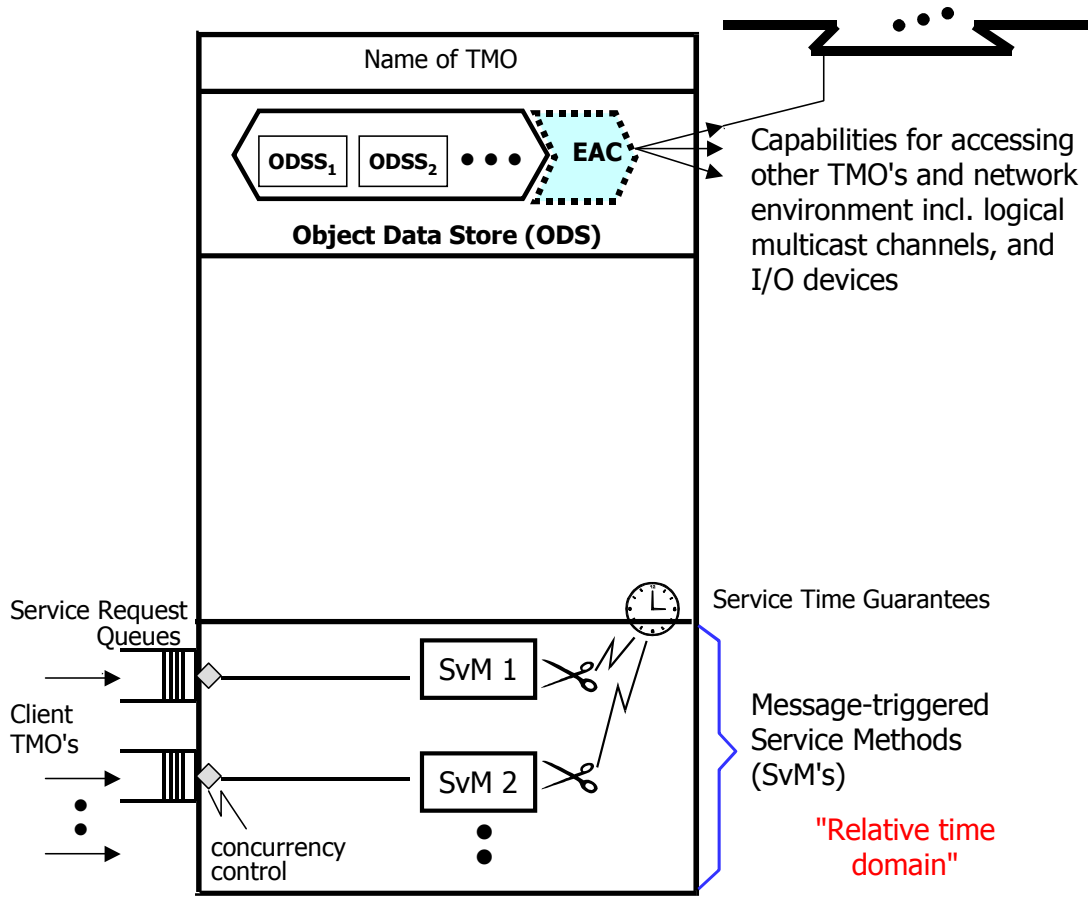
The TMO model

- Time-triggered (TT-) or spontaneous methods (SpM's):
- Time-window imposed on each output action and method completion
- Connections to the network environment as possible data members
- Basic concurrency constraint (BCC)



Environment Access Capability (EAC) section

- *Gate* objects providing efficient call-paths to remote object methods,
- I/O device interfaces, channel interfaces, etc;
- Each gate to a remote service method is associated with a guaranteed service time bound or a statistical service time guarantee;
- Client's call to a gate can be associated with deadline for result return, nothing more;



Spontaneous Methods (SpMs)

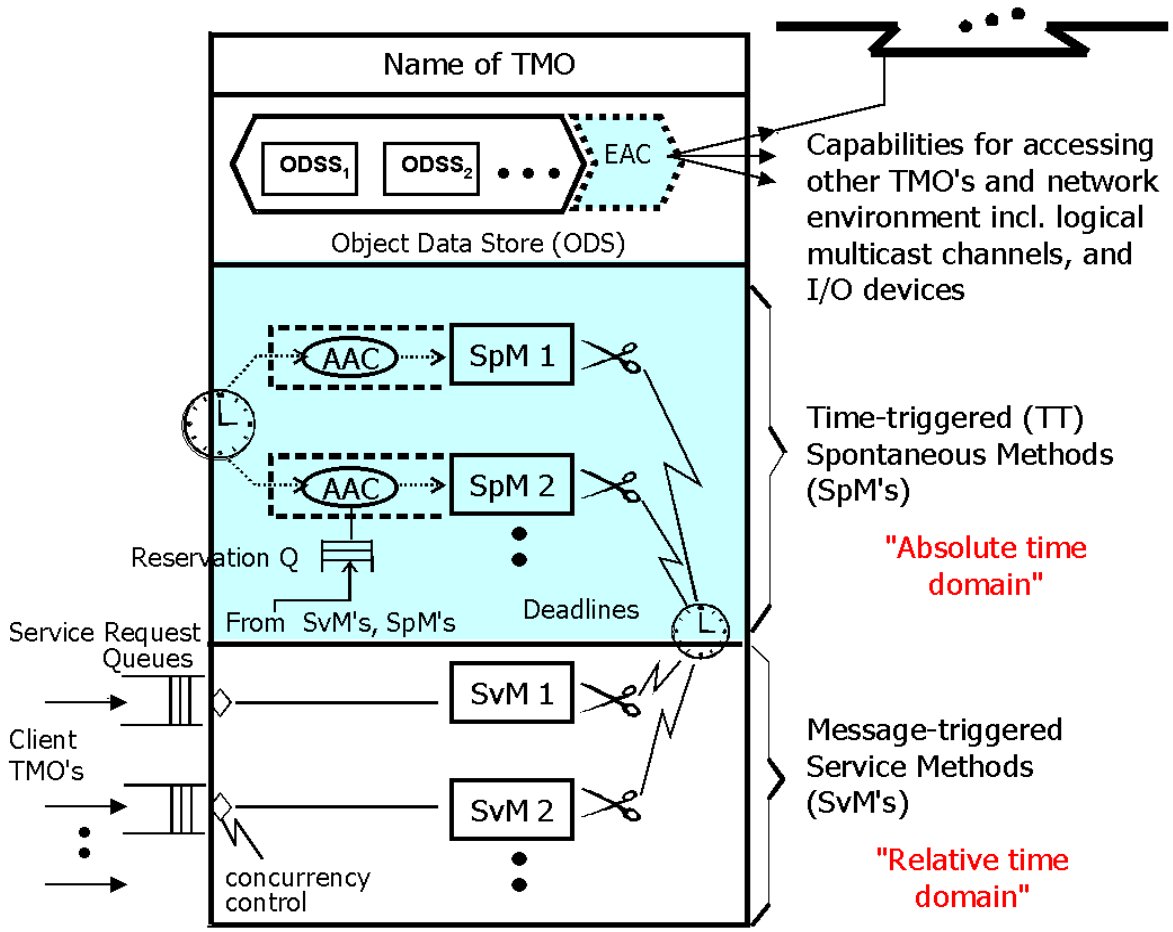
- **Time-triggered (TT-) or spontaneous methods (SpM's):**
  Clearly separated from the conventional service methods (SvM's) triggered by messages from clients

Example of AAC:

{"start-during (10am, 10:05am) finish-by 10:10am",

"start-during (10:30am, 10:35am) finish-by 10:40am"}.

Actions to be taken at real times determined at the design time appear only in SpM's



Autonomous Activation Condition

- Triggering times for SpM's
  - must be fully specified as constants at design-time
  - appear in the first clause of the SpM specification called the autonomous activation condition (AAC)

```
ab  "AAC-begin"
      { [AAC name:]
      "for t = from 10am to 10:50am every 30min
          start-during  (t, t+5 min)   finish-by  t+10min"
      }*
   ae "AAC-end "

==::

   {"start-during  (10am, 10:05am)       finish-by  10:10am",
    "start-during  (10:30am, 10:35am)   finish-by  10:40am"}.
```

Why TMO?

An application can be modeled as a network of distributed real-time objects

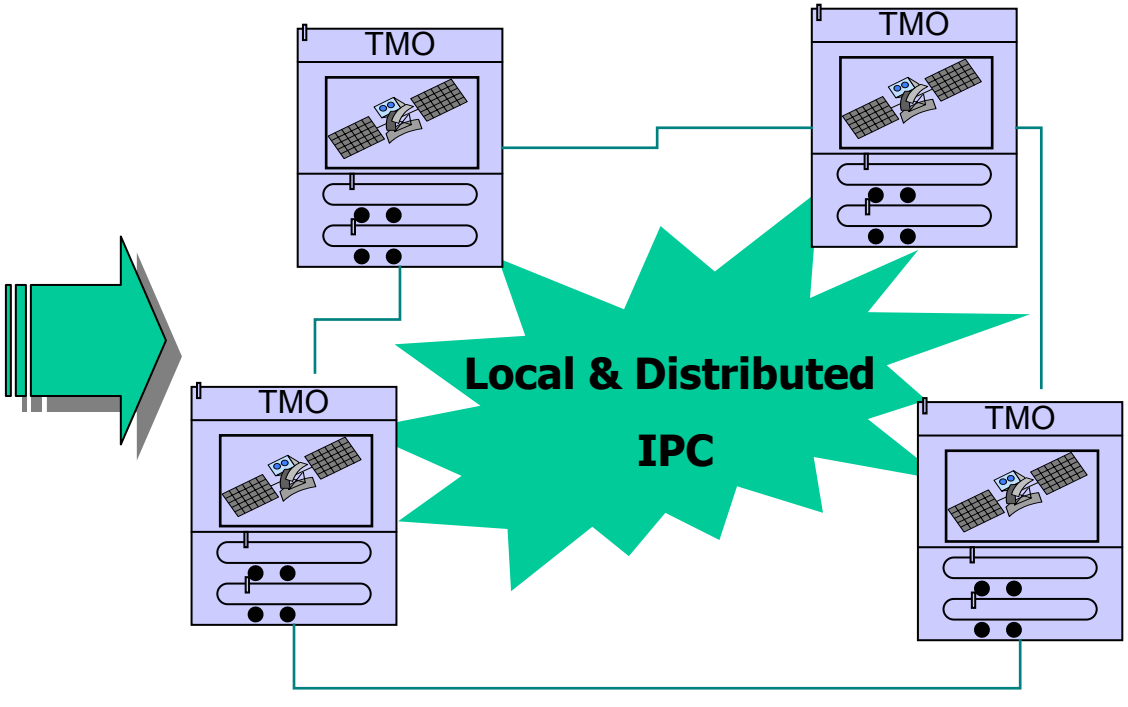
Concurrent Programming

Object Oriented Programming

Distributed Computing

Real-Time Computing

GUI



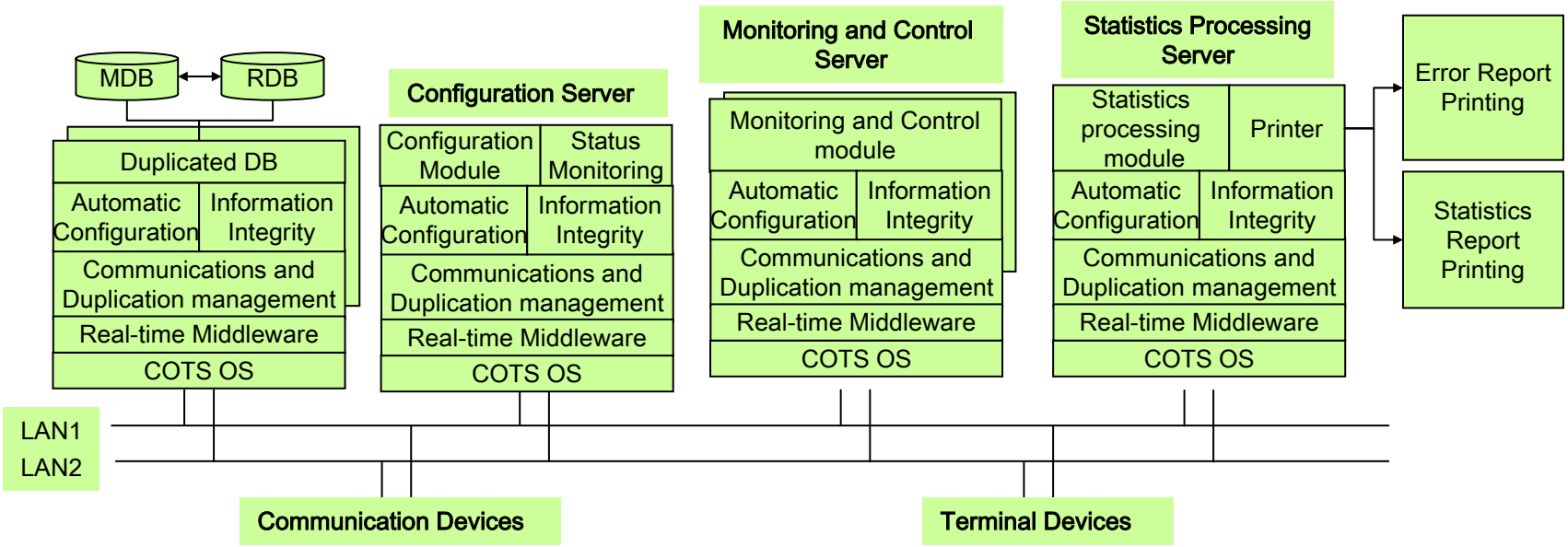
Distributed TMO Engines

Applications developed with the TMO structuring scheme

- Safety Injection System Simulator in a nuclear power plant [1999]
- Ship Radar Simulation System [2000]
- Wireless WebPad [2001] : BluePad
- MMI (Man machine Interface) [2002]
- Distributed War Game Simulation [2003]
- Prototype of a Robot Control System [2003]
- Oil Supertanker Control/Management System [On-going]

Hyundai Heavy Industries
Oil Supertanker Control/Management

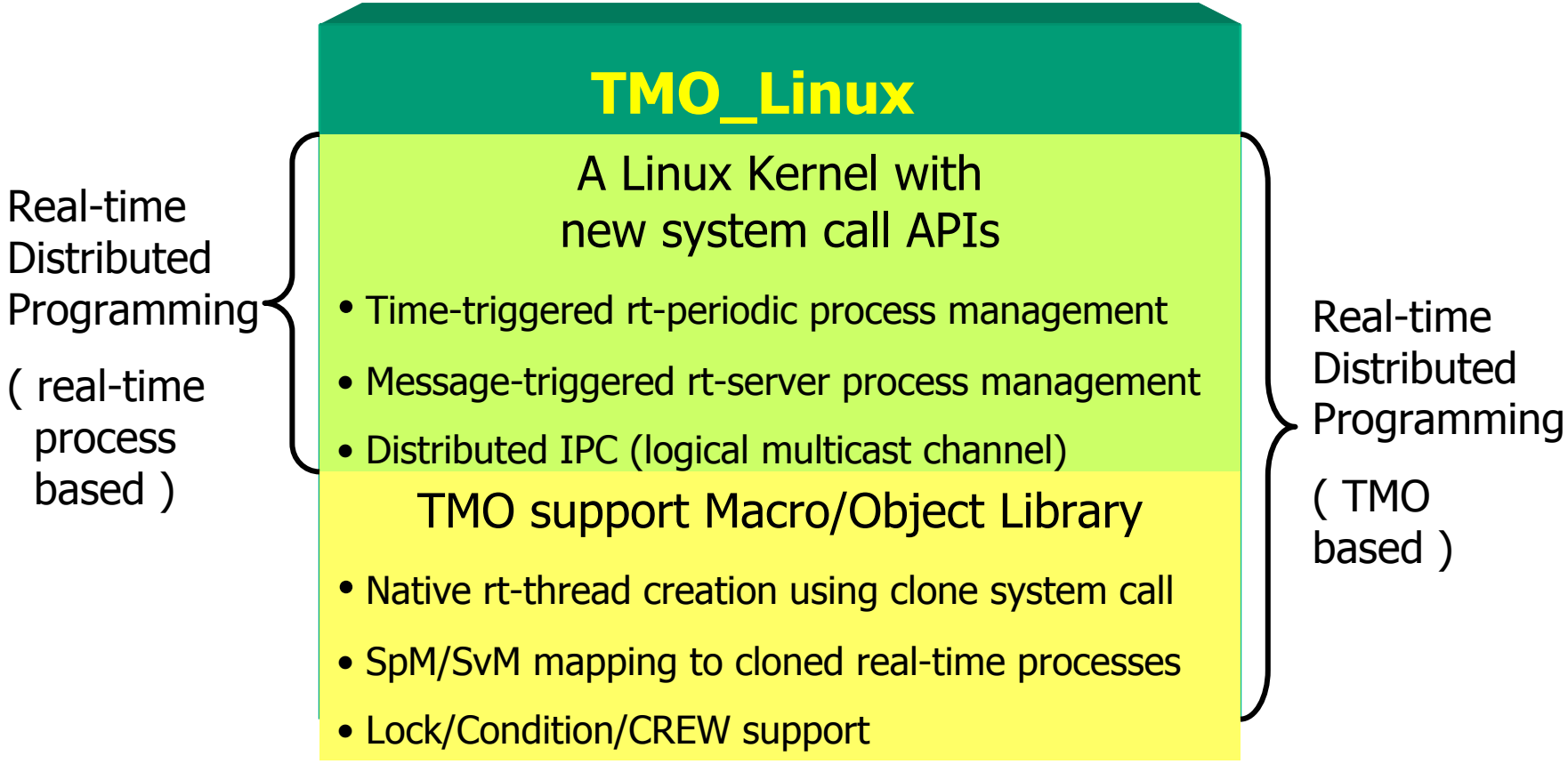
- Currently, developing a prototype of a configuration server by using TMO techniques



Support for TMO execution

- **Approach**
  - **OS: TMO-Linux**
  - **Middleware: TMOSM/Linux**
- **Basic features to support execution of TMOs**
  - **Scheduling**
    - On-time activation of SpMs
    - Deadline driven scheduling for SpMs and SvMs
    - BCC (Basic Concurrency Constraint): Activation of SvM is allowed only when potentially conflicting SpM executions are not in place
  - **Distributed IPC**
    - Network transparent logical multicast channels
  - **CREW/Lock**
    - Built-in Concurrent-read / Exclusive-write Monitor

Real-time Linux for server: TMO-Linux



- Both real-time middleware and TMO-Linux will have the same APIs as TMOSM/TMOSL developed at UCI DREAM Lab.

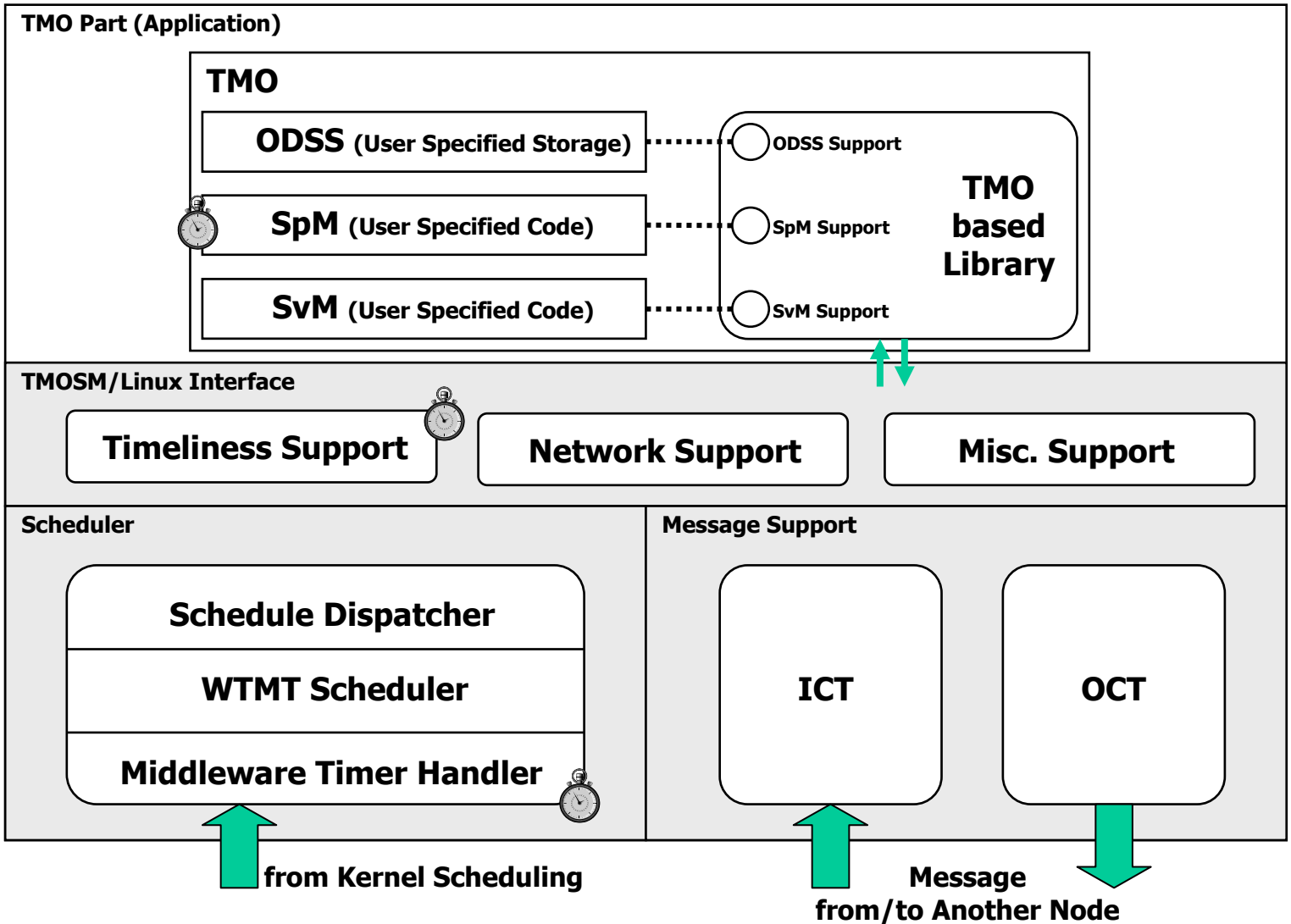
TMO-Linux

- **Supports (as system calls)**
  - **Real-time periodic processes/threads with deadlines**
    - *Rt-PP_Register* (period, deadline, start, stop);
    - *Rt-PP_Wait_Invocation* ()
    - *Rt-PP_Exit* ( );
  - **Network transparent distributed IPC**
    - *Alloc_Channel* (channel_ID, size_of_data, channel_type)
    - *Dealloc_Channel* (channel_ID)
    - *Send_Message* (channel_ID, &data)
    - *Receive_Message* (channel_ID, &data)
  - **Real-time server processes/threads activated by distributed IPC message with deadlines**
    - *Rt-SP_Register* (deadline);
    - *Rt-SP_Wait_Invocation* (channel_ID, msgp)
    - *Rt-SP_Exit* ( );
  - **Real-time scheduling (1/1000 sec. Resolution)**
    - Deadline driven
    - LLF (Least Laxity First)

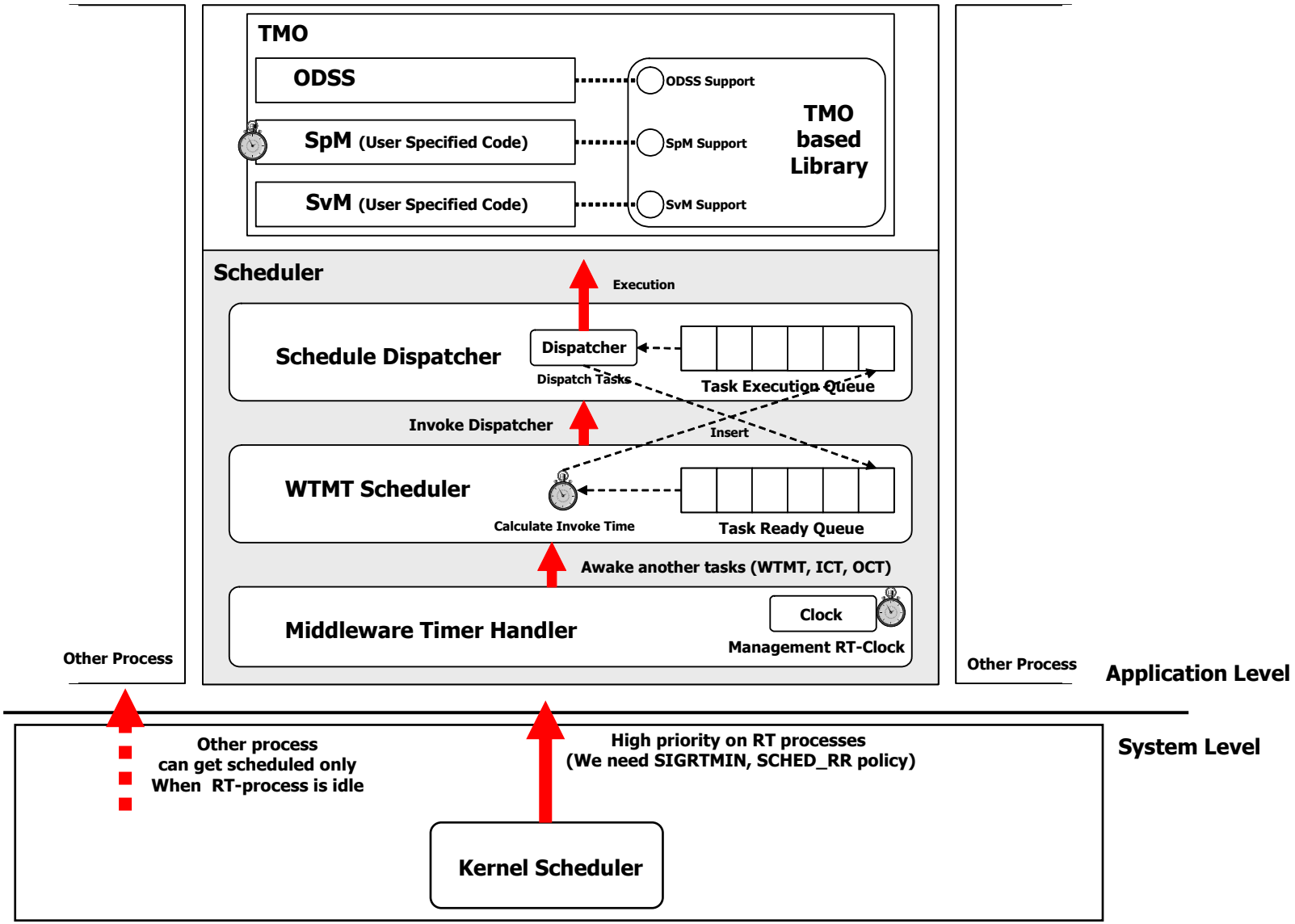
Real-time Middleware on Linux for Client

- KELIX
  - Konkuk Embedded Linux
  - Execute-In-Place (XIP) function
  - Main Memory Storage System
  - Optimized X Window
- Real-time Middleware (TMOSM/Linux)
  - Timeliness guarantee and deadline handler
  - TMO (Time-triggered Message-triggered Object) Model support
  - 10 msec clock resolution
  - POSIX.1b support
  - The same APIs as TMOSM/TMOSL developed at UCI DREAM Lab.

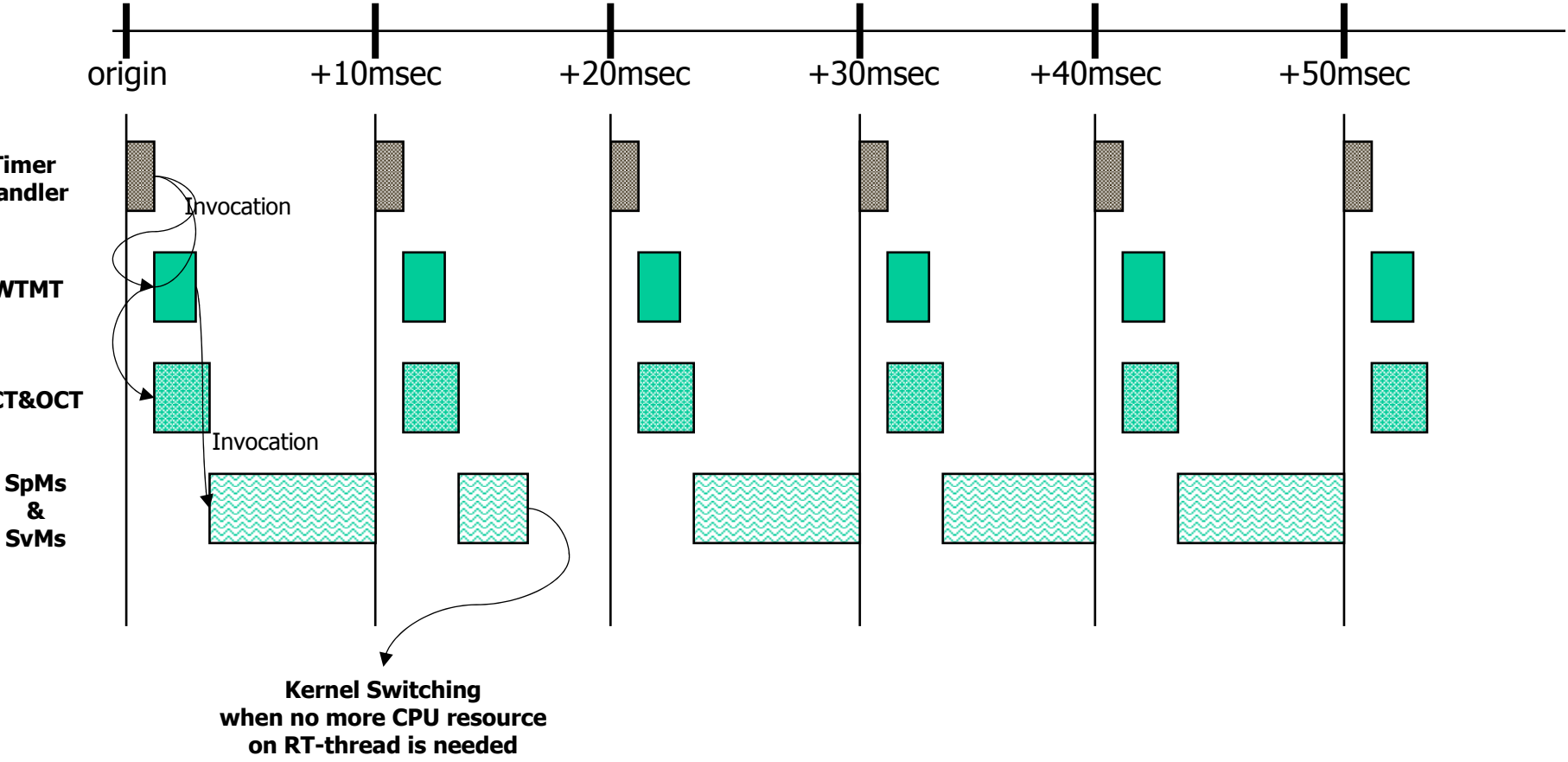
TMOSM/Linux


TMO Part (Application)
TMO
ODSS (User Specified Storage)
SpM (User Specified Code)
SvM (User Specified Code)
ODSS Support
SpM Support
SvM Support
TMO based Library
TMOSM/Linux Interface
Timeliness Support
Network Support
Misc. Support
Scheduler
Schedule Dispatcher
WTMT Scheduler
Middleware Timer Handler
Message Support
ICT
OCT
from Kernel Scheduling
Message from/to Another Node

TMOSM/Linux Scheduling



Timing Chart



Declaring a TMO

```
// Definition of TMO1
class TMO1 : public TMOBaseClass {
private:
   ODSSClass1   m_Odss1;
   TestSpM      m_SpM1;
   TestSvM      m_SvM1;
public:
   TMO1(const char* TMO_Name, TMOGateClass&gate1,
            const tms& TMO_start_time) :
      m_SpM1("SpM1", gate1, m_Odss1, RO),
      m_SvM1("SvM1", m_Odss1, RW)
  {
    // register with TMOSM/Linux
      activate(TMO_name, TMO_start_time);
  }
};
```

Declaring a ODSS

```
class ODSSClass1 : public ODSSBaseClass {
private:
  // Specific data structure defined by the TMO programmer
  int Temperature;
public:
   ODSSClass1() { Temperature = 0; }
   // ODSS functions defined by the TMO programmer
   void GetTemp(int *i) {
       EnterODSS_RO();
       *i= Temperature;
       ExitODSS_RO();
   }
   void SetTemp(int i) {
      EnterODSS_RW();
      Temperature = I;
      ExitODSS_RW();
   }
}
```

Declaring a SvM

- SvM Registration

```
// example: SvM constructor
   SvMClass1:: SvMClass1(const char* SvM_name,
        ODSSClass1& odss1,  access_mode_type mode1) :  m_pOdss1( &odss1)
   {
     build_regist_info_SvM_name(SvM_name); // register SvM Name
     build_regist_info_ODSS(m_pOdss1->get_id(), mode1); // register ODSS to access
     build_regist_info_max_invoke_rate(20*1000 ); // register MIR
     build_regist_info_guranteed_completion_time(20 * 1000); // register GST
     RegisterSvM( ); // register this SvM
   }
```

- SvMBody: implementation of service

```
void SvMClass1:: SvMBody()
{
     int     Client_RRQID;
  tmsp   timestamp;
     ReceiveSR(Client_RRQID, param, sizeof(*param), timestamp); // receive request
     // … do some real work
     ReplySR(Client_RRQID, param, sizeof(*param), timestamp); // reply to client
}
```

Declaring a SpM (1)

- **SpM Registration**

  Usually at the end of constructor of SpMClass, user calls RegisterSpM() to register SpM

```
// example: SpM Constructor
SpMClass1:: SpMClass1(TMOGateClass& gate1, // pass gate reference
                      ODSSClass1& odss1,    // pass odss pointer
                      access_mode_type mode1) :
                    m_pGate1( &gate1), m_pOdss1( &odss1)
{
  build_regist_info_ODSS(m_pOdss1->get_id(),mode1); // define ODSS to access
  MicroSec   from  =            5;   from  *= 1000 * 1000;
  MicroSec   until = 2 * 60 * 60;   until *= 1000 * 1000;
  MicroSec   every = 1 * 1000 * 1000;
  MicroSec   est = 0;
  MicroSec   lst = 5 * 1000;
  MicroSec   by = 100 * 1000;
  AACclass AAC1(tm4_DCS_age(from), tm4_DCS_age(until),
                    every, est, lst, by,""); // create an AAC instance
  build_regist_info_AAC(AAC1); // register AAC
  RegisterSpM( ); // register SpM
}
```

Declaring a SpM (2)

- **SpMBody**

  Implementation of SpM task. A very simple example would be like, (The scenario is that this SpM gets some data from other TMO's, then updates the content of ODSS using the newest data)

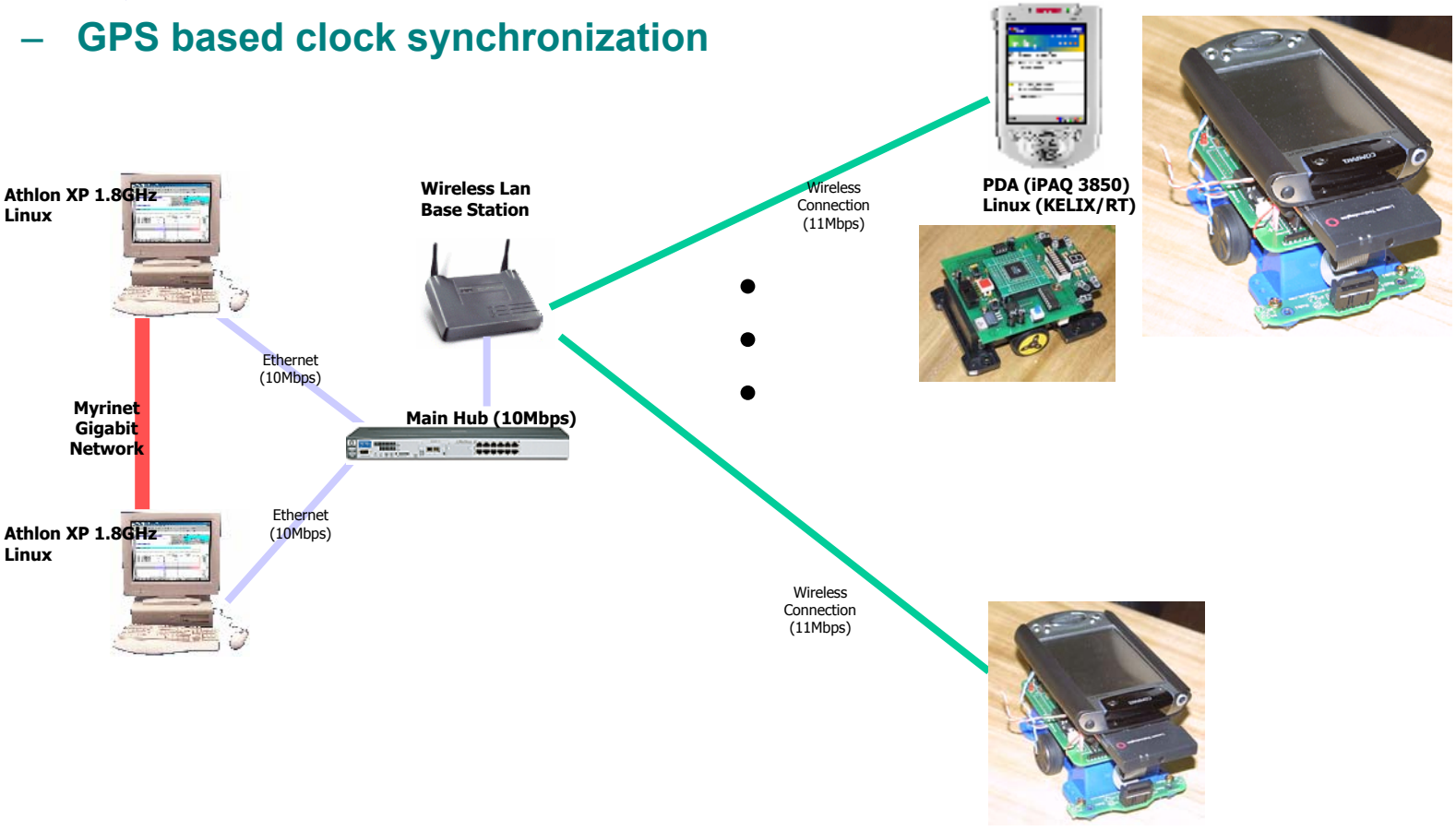
```
void SpM1Class:: SpMBody( )
{
        // define local variables;
        tmsp timestamp;
        ParamStruct SvMPara;

        // call service from other TMO's, return data is stored in SvMPara;
        m_pGate1->BlockingSR(&SvMPara, sizeof(SvMPara),
           timestamp);

        // update ODSS; Update(ParamStruct  *) is a method of ODSS
           class;
        m_pOdss1->Update(&SvMPara);
}
```

Application initialization

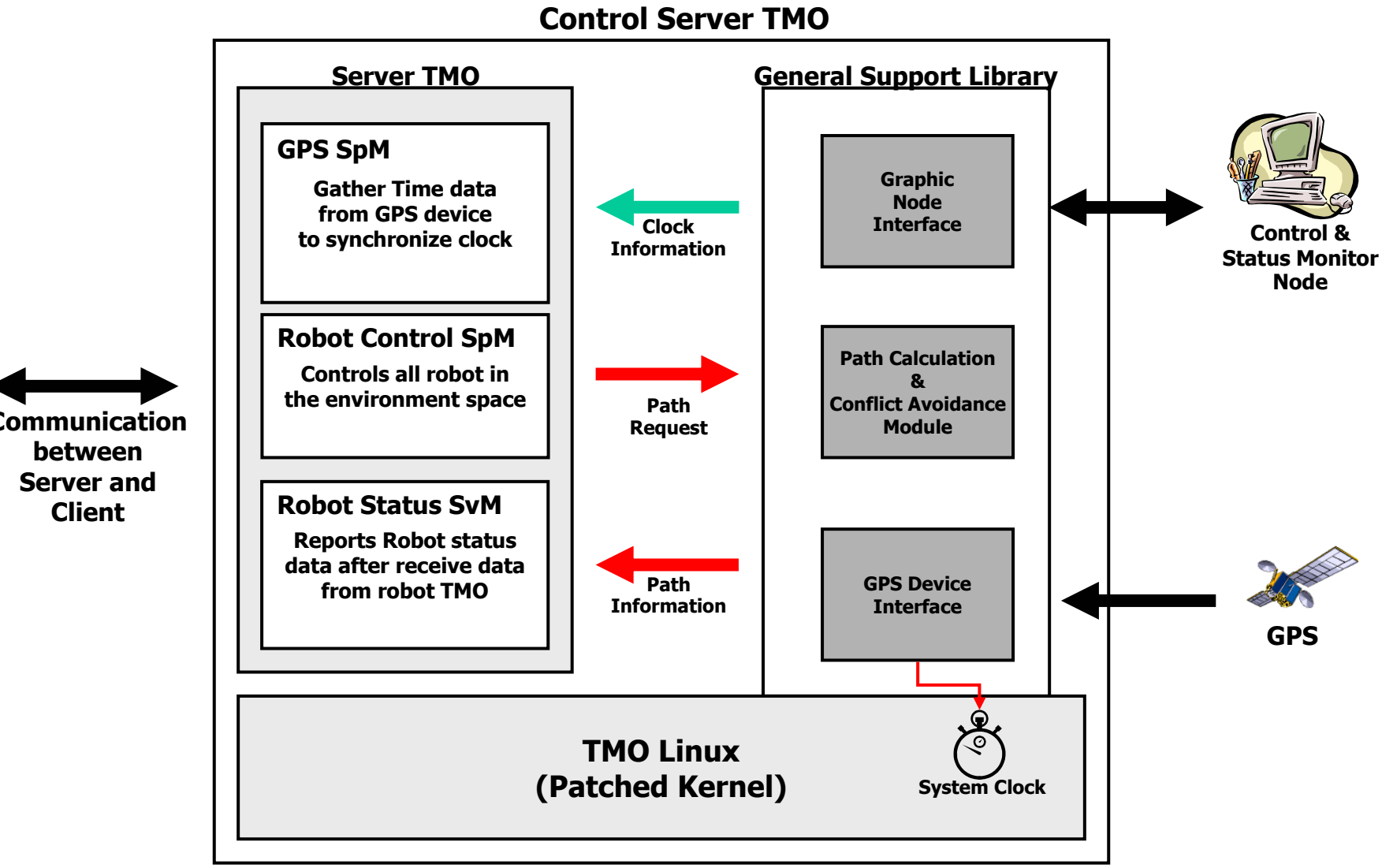
```
void main( )
{
    // TMOSM should be started before doing anything else
    StartTMOEngine( );

    // Define a TMO gate class
    TMOGateClass gate1("TMO2", "SvM1", tm4_DCS_age(50*1000*1000));

    // Define a TMO start time
    tms TMO_start_time1 = tm4_DCS_age(5*1000*1000);

    // Instantiate a TMO.  It is registered to TMOSM in its constructor
    TMOClass1 T1("TMO1", gate1, TMO_start_time1);

    // Main thread goes to sleep
    MainThrSleep();
}
```

Test-bed

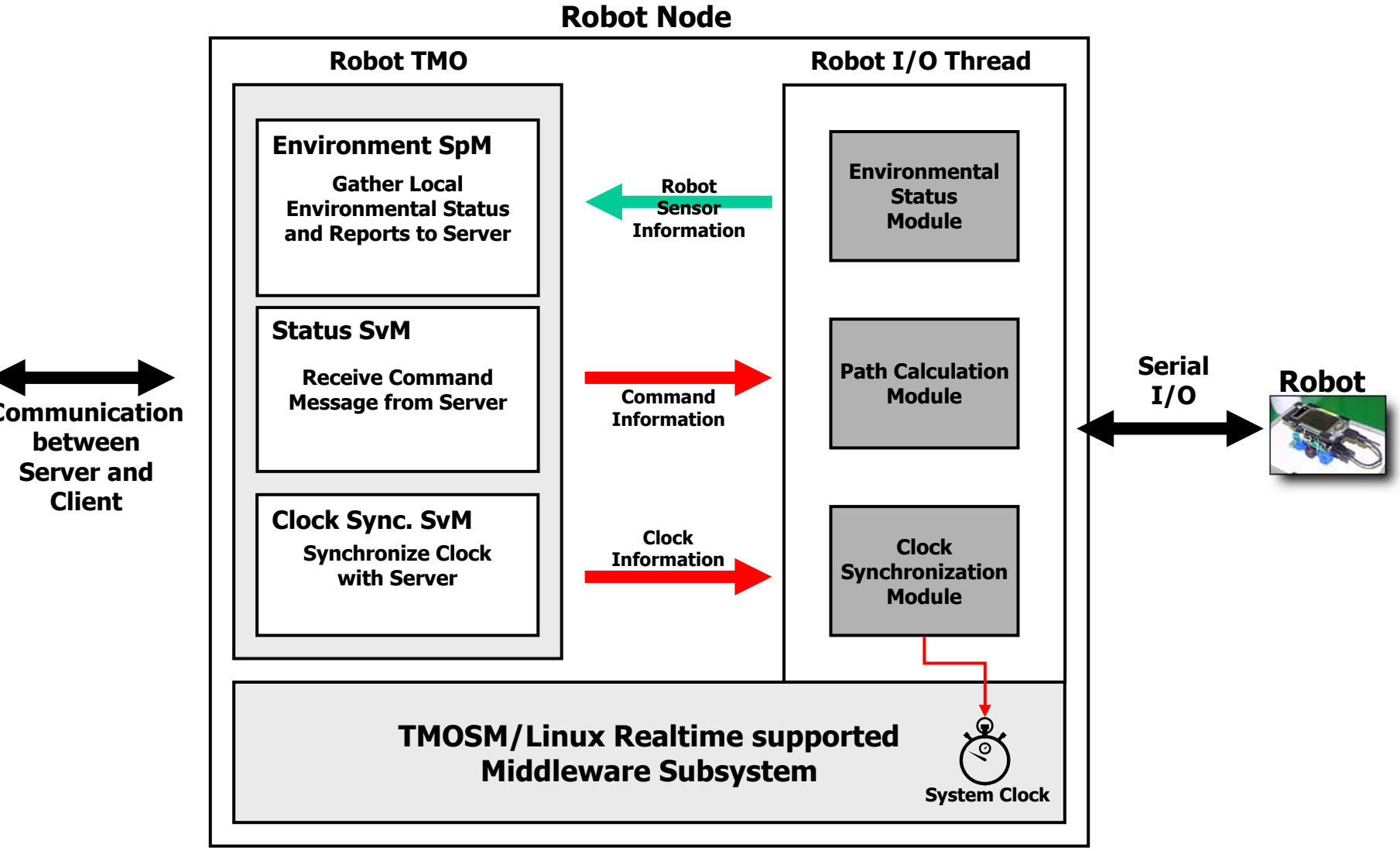
- Unmanned Airplane Control System
- 2 PCs, 2-5 robots with PDAs
  - GPS based clock synchronization



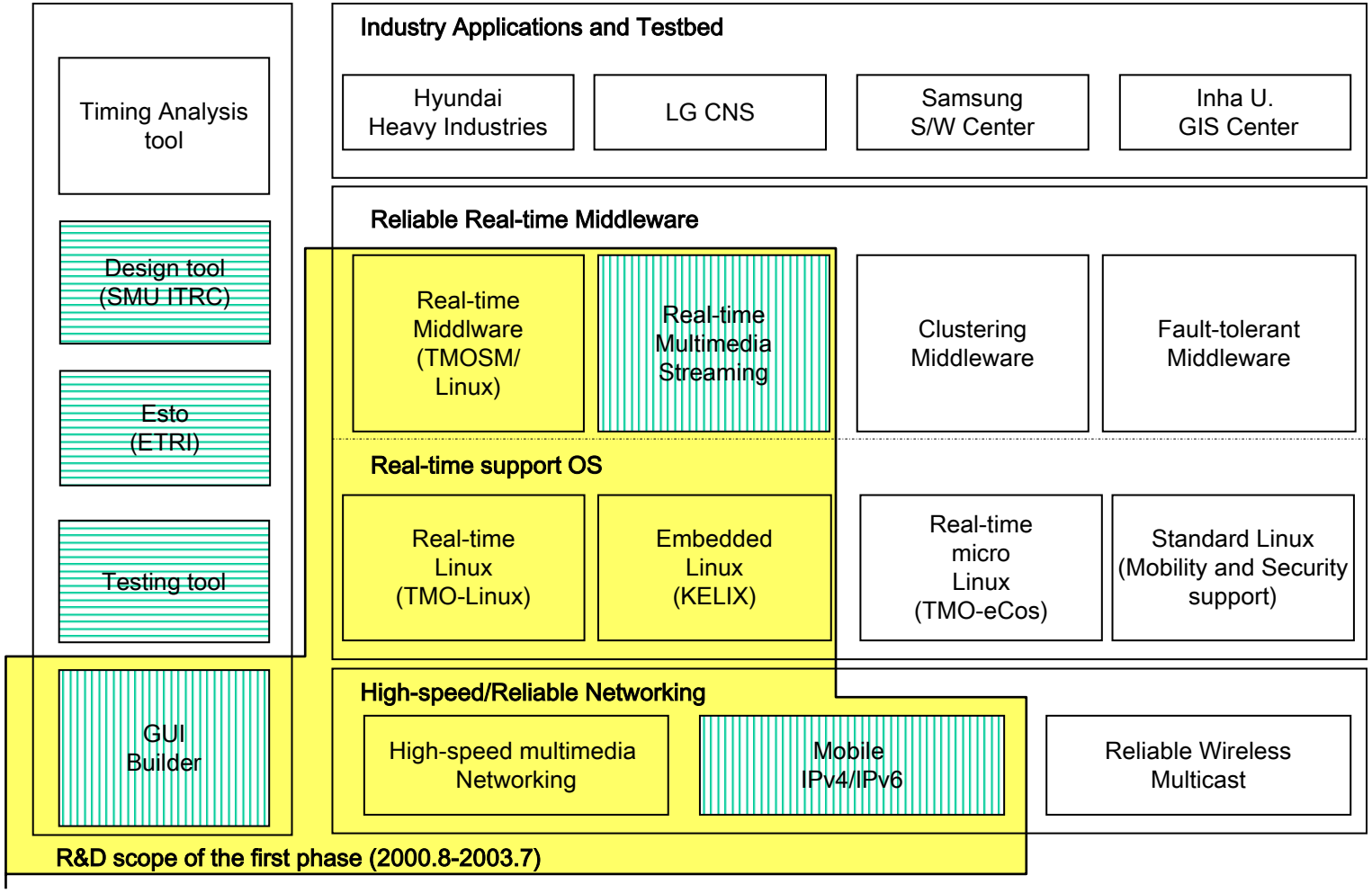
TMO Structure (Server)



TMO Structure (Client)



Future Plan (2003.8 – 2007.7)



Introduction to ESTC Projects

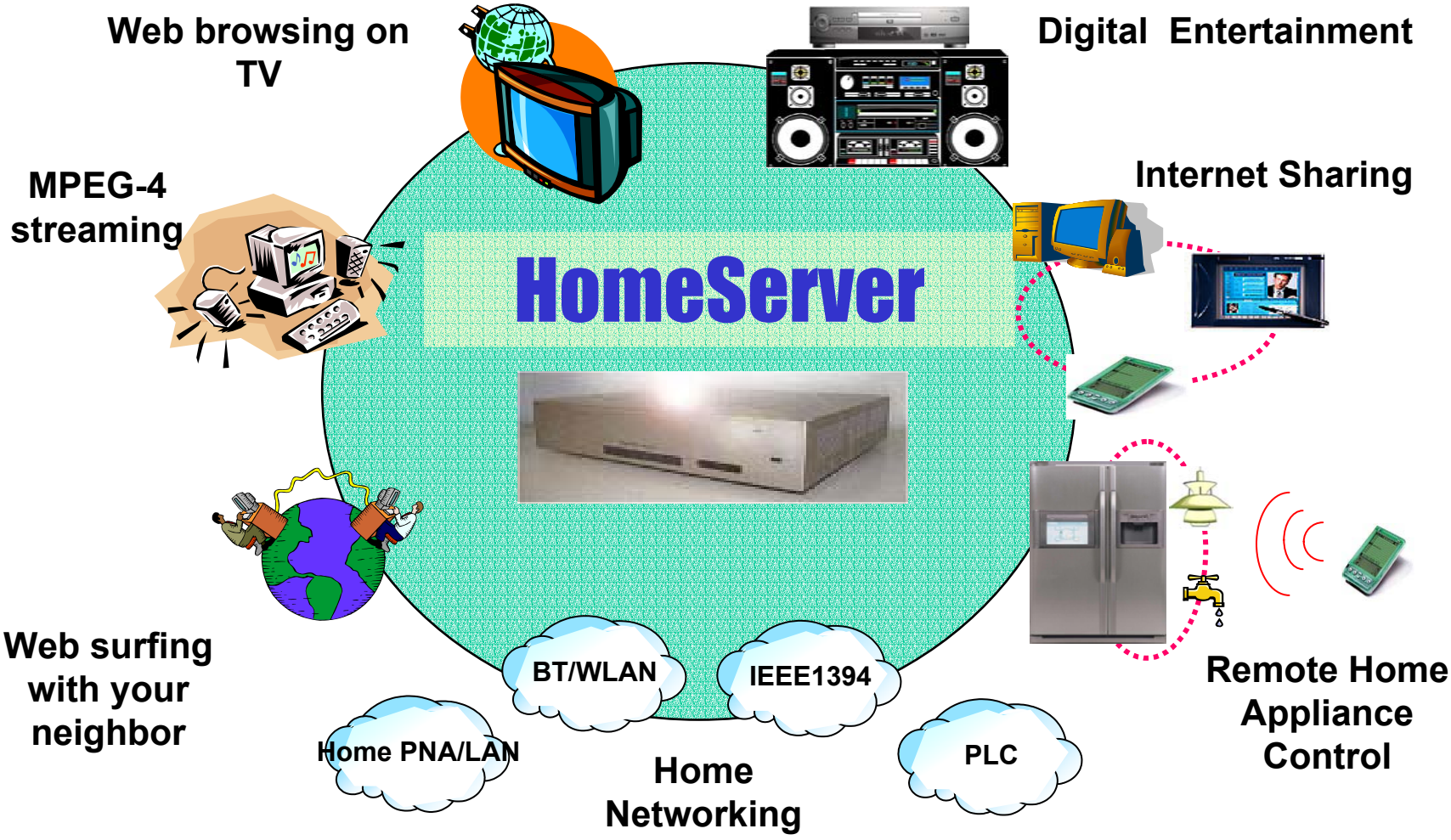
Embedded S/W Technology Center(ESTC)

Electronics and Telecommunications Research Institute

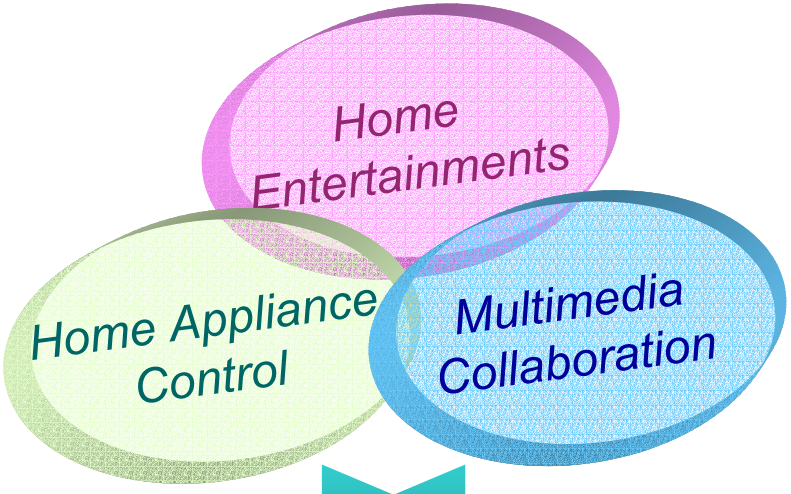
Embedded S/W Technology Center (ESTC)

- Supported by Ministry of Information and Comm.
- http://www.etri.re.kr/english/comso/
- Objectives
  - To develop efficient national information infrastructures
  - To enhance international competitiveness
  - To transfer technology to private/public corporations
- Research areas
  - Digital Home (Home server & networking)
  - Real-time OS and middleware
  - Portable client platform.
- Current activities
  - Development of embedded software standard platform technology (2003 ~2005)
  - Development of digital home technology (2002 ~ 2005)
- Research staff: 90+

Home Server and Digital Home



Current Research of ESTC in ETRI



Embedded Linux Development Solutions including

OS & IDE

Qplus

Esto

Home Entertainments

- DVD
- Java-based Web Browser (named WebJABI)
- Java-based eBook Reader
- MP3 (Karaoke)
- Video On Demand (MPEG1/2, WMV)
- UPnP through OSGi
- SMS
- DTV

Home Appliance Control Services

Middleware independent
Home Appliance Control Service

Integrated Control Middleware (CM)

Jini
for Data network

HAVi
for AV Network

LonWorks
for Control Net.

Various Home Appliances

Collaboration beyond DTV

Interactive Multimedia Collaboration Services

- Collaboration Services on Home Servers
  - Video Phone, Web-based collaborative Browsing
- Video Surveillance
  - Internal
  - External
  - Record and play
- Mobile Collaboration
  - VoIP, Messenger, White Board
- Streaming Service between Home Servers
  - Authoring Tool, MPEG-1, MPEG-4

Qplus Embedded Linux

- Qplus
  - Embedded Linux Solution for Digital Devices
- Qplus Components
  - Linux Kernel 2.4.x with Lock Break Patch
    (New real-time and Low Power Management features to be added)
  - Fast Booting for x86 Architecture
    - Much Faster than Other Similar Architectures (3 times)
  - Multimedia File System
  - GUI Libraries : TinyX/GTK, QT/Embedded, …
  - System Applications : Busybox, Tinylogin, Boa, …
  - Multimedia Applications : MPEG1-2-4, DVD, MP3 player, …
  - Java Applications : Web Brower, E-Book
- Foot-print
  - Start from 2MByte ~

Real-time Features

- Preemptive kernel with Lock Break Patch
- Optimizing performance for our platforms
  - Optimize device drivers
- Sporadic Server (POSIX)
  - Kernel Level Implementation
- Qos based File System Management for multimedia
  - Qos access (open/read/write) System Calls are added
  - Qos open reserves the proportional bandwidth capacity needed for this access
  - Qos access is performed with a deadline (timeout)

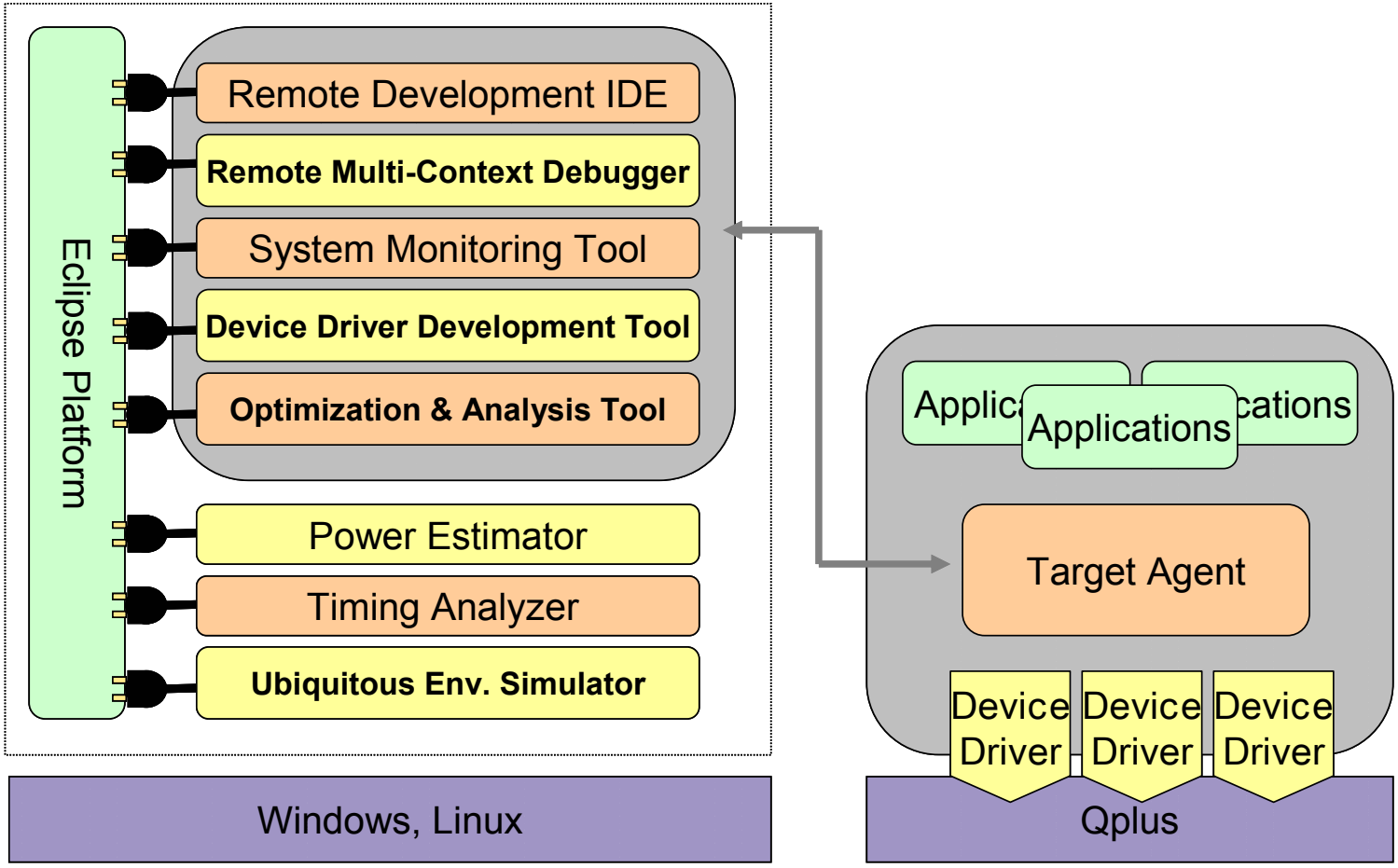
Esto – Embedded Linux Development Toolkit

- Accelerate developer's productivity of embedded software development based on Qplus
- Support Platforms
  - Target : x86, ARM9, MIPS, SMDK
  - Host : Linux, Windows
- Features
  - Rich Tools : Target Builder, IDE, Debugger, Power Estimator, System Monitor/Tracer, Target Agent, Host Agent
  - User Friendly GUI : Simple to use, Easy to learn, Same look and feel on both Linux and Windows hosts
  - Embedded Software Tailored Feature : Measure power consuming, Check real time constraint, Trace point based non-stop debugging

Target Builder :
Qplus Configuration & Embedding Tool

- **The First CML (Configuration Meta Language) 2 based Unified Configuration System**
  - You can configure kernel, packages and target specific options altogether
  - Dependencies are checked automatically
- **Create Embedded Linux System in just a few minutes.**
  - Just load provided pre-configurations for each BSP
  - Point & Click selection of OS component with easy to use GUI interface
  - Deploy to the target simply by clicking a ‘Deploy’ button
- **Create very optimized system ROM image**
  - Fine-grain package control: file list, compile options, …
  - Library optimizer : both file and symbol level

Esto Plug-in for Eclipse



Summary

- **Works on KKU SWRC and ETRI ESTS**
  - **Object-oriented paradigm support**
  - **Timeliness is emphasized**
  - **Based on open source software (mainly, Linux)**
  - **Towards ubiquitous computing**
- **Most activities by others have similar characteristics**
  - **Object-oriented or component-based approach**
  - **Towards ubiquitous computing**
    - Security, mobility, and reliability are key issues.
    - Intelligent HCI mechanisms are important.