Control Basado en Misiones

DPI2011-28507-C02-01/02

Informe interno

T0302:
Desarrollo del middleware de gestión de agentes

Autor:	**Eduardo Munera, Juan Luis Posadas, José Luis Poza, Manuel Muñoz, Juan Francisco Blanes.**
Revisor:	**José Simó**
Fecha:	
Resumen:	

UNIVERSIDAD POLITECNICA DE VALENCIA

CONTENIDO

1. Introducción al Control Kernel Middleware

El Control Kernel Middleware CKM (o Núcleo de Control) ofrece el soporte necesario para la ejecución de tareas de control en sistemas empotrados con recursos limitados. Tal y como se describe en el Hito 3a el CKM representa el principal componente de la arquitectura software de los agentes, ya que proporcionará los mecanismos básicos para ejecutar las tareas de control requeridas por las misión de control asignada al agente.

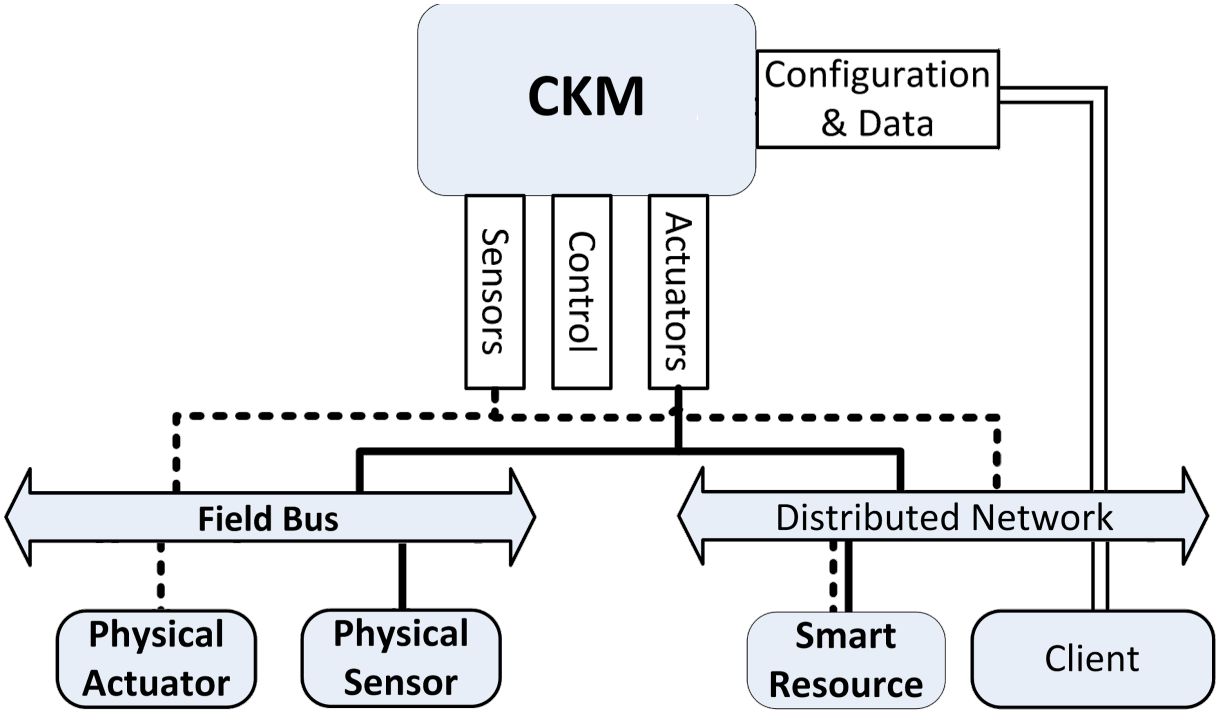

Figura 1. Esquema del Control Kernel Middleware

En la Figura 1 se representan los principales componentes de la implementación del CKM así como sus capacidades básicas de interacción, destacando los siguientes elementos:

- Tareas de Control: El CKM está diseñado para gestionar y ejecutar las tareas típicas de un sistema de control, a saber: control, sensorización y actuación. Para ello dispone de un planificador de tiempo real capaz de garantizar el periodo de ejecución de cada una de las tareas. Para el caso de las tareas de sensorización y actuación garantizará acceso a los diversos dispositivos entrada y salida del sistema así como ofrecerá comunicaciones a través de buses de campo. Adicionalmente el sistema permitirá la ejecución de tareas (periódicas o esporádicas) que no estén relacionadas directamente con el objetivo de control del sistema.

- Gestión de actuadores y sensores: El CKM garantiza el acceso y configuración de los diversos periféricos hardware acorde al dispositivo que lo implementa. Para

ello utilizará una lista de canales disponibles, los cuales serán estrictamente dependiente de la arquitectura y hardware utilizado en cada caso.

- Buses de Campo: Para la comunicación con dispositivos básicos como sensores o actuadores que lo requieran, así como con otras dispositivos que implementen el CKM y no dispongan una capa TCP sobre la cual proporcionar servicios como Smart Resources

- Comunicaciones Distribuida: Aquellos dispositivos con un interfaz de comunicaciones que implementan TCP tendrá la capacidad de ampliar la especificación del middleware incorporando un sistema de comunicaciones basado en el paradigma publicación/subscripción. Los dispositivos que dispongan de estas capacidades podrán ser usados como soporte físico para la implementación de Smart Resources (Hito 3 a).

Dadas las capacidades del HW utilizado, así como las necesidades del sistema existen dos niveles de implementación claramente diferenciadas:

- Tiny CKM: Esta es la implementación mínima del middleware, concebida para su aplicación en dispositivos con recursos limitados en los cuales se requiere efectuar tareas de control de bajo nivel y no requieren de comunicaciones de alto nivel, ni reconfiguraciones frecuente de su funcionamiento.

- Full CKM: Amplia las funcionalidades incluyendo un middleware de comunicaciones para gestionar la configuración del dispositivo, además de ofrecer datos procesados de alto nivel y permitir la monitorización de las medidas de calidad y desempeño del sistema.

2. Middleware de comunicaciones

El middleware tiene la capacidad de funcionar sobre un sistema distribuido donde la comunicación cubre aspectos de diversa complejidad como son una correcta conexión y direccionamiento de los mensajes. Por este motivo, es necesario disponer de un sistema de comunicación que garantice la entrega fiable de mensajes. En este sentido se ha implementado uno propio basado en el modelo publicación/suscripción para este propósito. El modelo publicación/suscripción permite el envío y recepción de datos, eventos y comandos entre los nodos del sistema mediante el uso de topics. En este contexto, los publicadores son responsables de la escritura de valores en el topic mientras que los suscriptores leen de los mismos.
El sistema de comunicaciones implementado gestiona las conexiones necesarias para difundir mensajes entre los peers de la red, de modo que se establece una conexión bidireccional entre los diferentes pares de nodos. En este modelo, todos los peers conocen la existencia de los otros pares conectados por lo que estamos hablando de una topología de malla totalmente conectada. Cada peer dispone de un identificador único que difunde a través del canal al resto de nodos de la red. Estos identificadores están compuestos por la

dirección IP del dispositivo donde reside y su puerto. En la siguiente figura se muestra una topología de los peers.

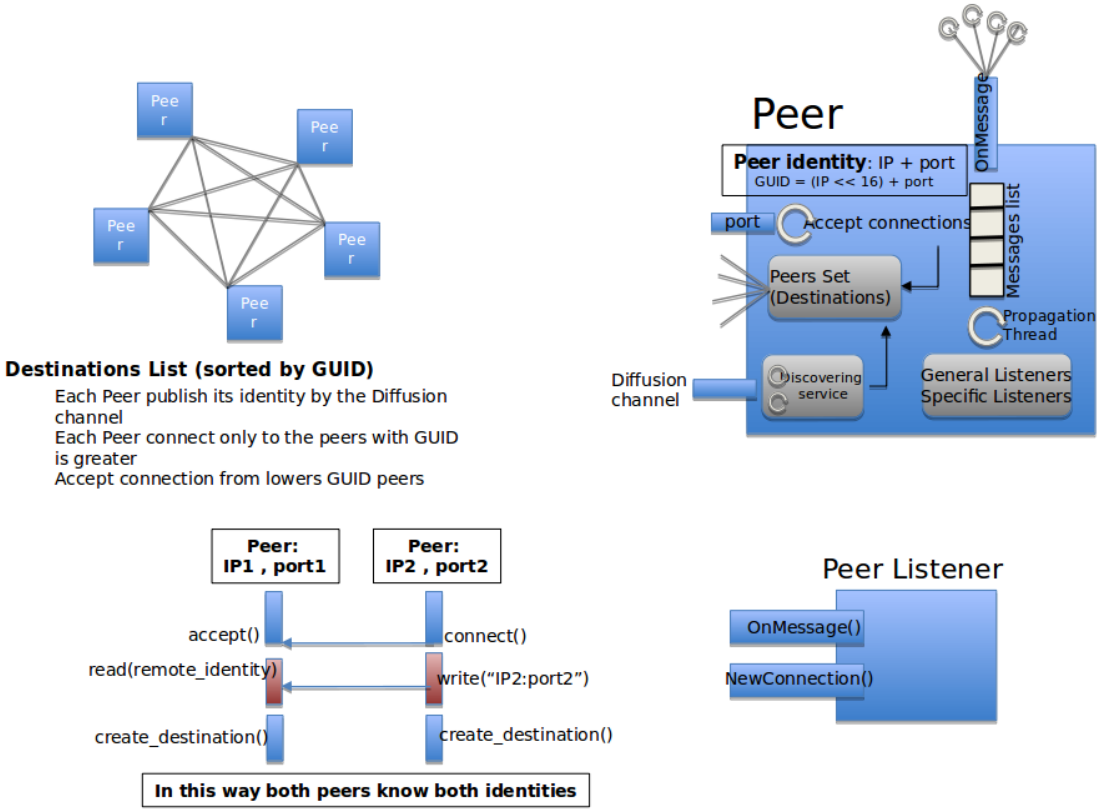

Figura 3. Descripción de los peers del sistema.

Con el fin de optimizar las comunicaciones y reducir el tráfico duplicado en la red, cada par solo conecta con aquellos nodos que tengan un identificador superior al suyo propio. De esta forma aparece el concepto de nodo máster encargado de enrutar los mensajes provenientes de la red a los nodos correspondientes. En caso de trabajar con diferentes redes, un peer a su vez actuará de proxy para el intercambio de mensajes. El intercambio de mensajes entre peers será diferente naturaleza en función de la acción a tomar, ya sea por la suscripción a topic, mensajes de sincronización, etc. En la figura 4 se muestra los tipos de mensajes soportados así como su formato.

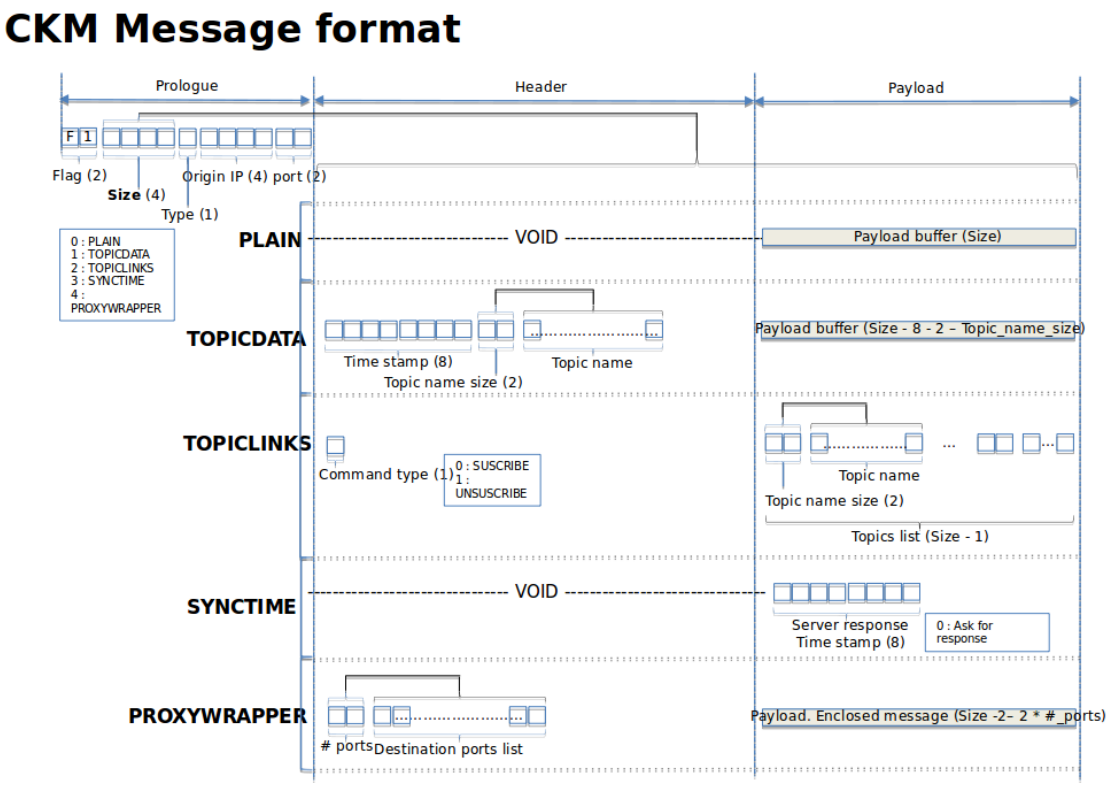

Figura 3. Tipos de mensajes usados en el Middleware de comunicaciones y estructura de los mismos.

El middleware de comunicaciones ofrece las funcionalidades necesarias para el intercambio de información del middleware de control con otros nodos del sistema. A través de la publicación en topics, los tipos de datos asociados que se pueden encontrar son:

- Parámetros de configuración
- Medidas de calidad y desempeño del sistemas
- Información de alto nivel asociadas al servicio proporcionado
- Otros parámetros de monitorización

3. Compilación del CKM para diversas arquitecturas

Puesto que la implementación del CKM es estrictamente dependiente del hardware usado como soporte de físico de ejecución el código desarrollado ha sido cuidadosamente dividido según las partes dependientes o no a la plataforma. Tal y como se observa en la Figura 4, la estructura software del CKM ha sido establecida en dos partes básicas: el código común a cada plataforma, y las secciones relativas a la gestión y acceso del hardware (Hardware access layer HAL). Como resultado el mismo código podrá ser compilado mediante IDEs diferentes para obtener versiones compiladas para diferentes plataformas y arquitecturas. Para cada una de las arquitecturas se incluye un set de tests de ejemplo a fin de testear el correcto funcionamiento del CKM. Las plataformas que disponen de implementación para el CKM son:

- Microchip:
  - pic 32 bits
  - pic 16 bits
- v-rep (simulador de robótica)
- Linux (sin acceso a GPIO)

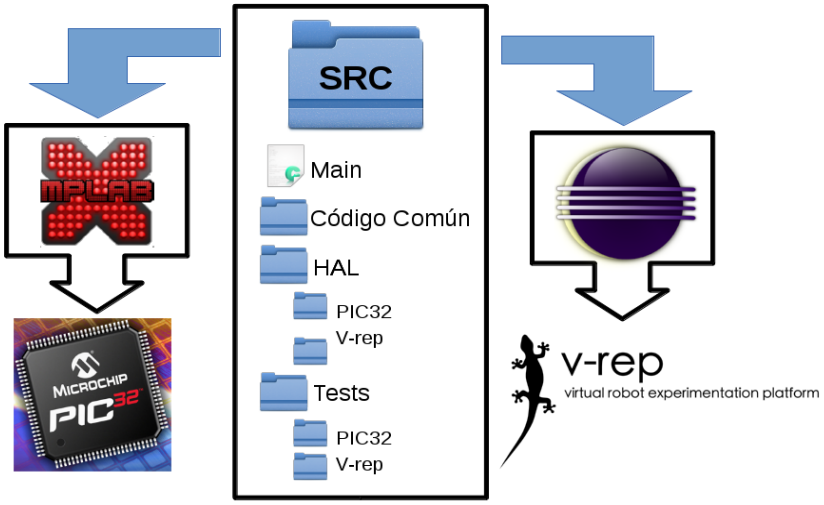

Figura 4. Ejemplo de compilación de código para múltiples plataformas y arquitecturas.

4. Sistema distribuido basado en CKM Smart Devices

Gracias a las capacidades de comunicación de las implementaciones basadas en Full CKM se puede establecer una red de dispositivos distribuidos cuya ejecución recae en el funcionamiento en las funciones de control del CKM tal y como se puede apreciar en la Figura 5.

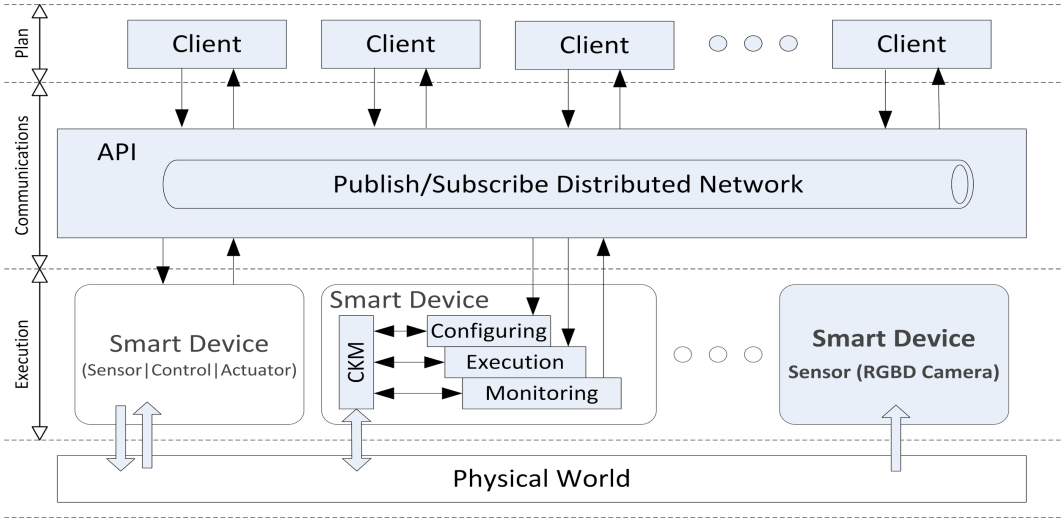

Figura 5. Sistema distribuido formado por Smart Devices con ejecución basada en el CKM.

5. - Anexo: API CKM

Tareas de control

- Sensorización

```
#define CH_N X
#define TOPIC_S X
#define PERIOD_MS X
#define PERIOD_US X
SMSRV_sensor *sensor;
KM_sensor_configuration conf;

SMSRV_get_configuration (CH_N,TOPIC_S,PERIOD_MS,PERIOD_US,
&conf);
SMSRV_sensor_create (TOPIC_S, conf, &sensor
SMSRV_sensor_delete (&sensor);
```

- Actuación

```
#define CH_N X
#define TOPIC_A X
#define PERIOD_MS X
#define PERIOD_US X
ACTSRV_actuator *actuator;
KM_actuator_configuration conf;

ACTSRV_get_configuration(CH_N,TOPIC_A,PERIOD_MS,PERIOD_US,
&conf);
ACTSRV_actuator_create (TOPIC_A, conf, &actuator);
ACTSRV_actuator_delete (&actuator);
```

- Control

```
#define TOPIC_R X
#define PERIOD_MS X
#define PERIOD_US X
CTRLSRV_regulator *regulator;
KM_regulator_configuration conf;
void ctrl_function (CTRLSRV_regulator *reg);
```

```
CTRLSRV_get_configuration(TOPIC_S,TOPIC_R, TOPIC_A, PERIOD_MS,
PERIOD_US, &ctrl_function, &conf );
CTRLSRV_set_PID (P, I, D, &conf); (opcional)
CTRLSRV_regulator_create (conf, &regulator);
void ctrl_function (CTRLSRV_regulator *reg){
        //Lectura de entrada y referencia
        X = regulator->in_data.information
        X = regulator->ref_data.information

        //P – I – D
        regulator->conf.controler_params[0] //P
        regulator->conf.controler_params[1] //I
        regulator->conf.controler_params[2] //D

        //Escritura en la salida
        regulator->out_data.information = X;
}

CTRLSRV_regulator_delete (&regulator);
```

Tareas de adicionales

- Tarea ciclica

```
#define T X
#define PRIO X
#define ID X

SCHEDSRV_CSActivity *act;
SCHEDSRV_Cyclic_Scheduler *sched;
void act_fnc (SCHEDSRV_CSActivity *act);

SCHEDSRV_new_scheduler_config( PRIO_SCHED, &sched );(Si no existe)
SCHEDSRV_new_activity_config (shced, T, PRIO, ID, &act_fnc, &act);

SCHEDSRV_activity_delete (scheduler, &act )
```

- Tarea esporádica

```
#define T X
#define PRIO X
#define ID X

SCHEDSRV_EActivity *act;
```

```
void act_fnc (SCHEDSRV_EActivity *act);
SCHEDSRV_new_esporadic_activity (ID, &act_fnc, &act);
```

Gestión de comunicaciones

```
#define CH_N X
MSGMNGR_resource *handler;

KM_openComm_API(CH_N, &handler);
void *data;

API_dataComm_output(data,handler);
API_dataComm_input (data,handler);

KM_closeComm_API(&handler);
```

Gestión de canales de entrada y salida desvinculados de tareas de sens/act

```
#define CH_N X
IOSRV_struct *handler;

KM_openChannel_API(CH_N, &handler);
void *data;
int size;

API_data_output(data,size,handler);
API_data_input (data,size,handler);
KM_closeChannel_API(&handler);
```

6. - Anexo API Middleware Comunicaciones

Inicialización

```
WMP_init(port);
```

Suscripción

```
Topic t1;
strcpy(&(t1.name[0]),"T01");
t1.s_name = 3;
WMP_addListener(&t1);
```

Publicación

```
Topic t1;
strcpy(&(t1.name[0]),"T01");
t1.s_name = 3;
strcpy(&(t1.value[0]),"0123456789");
t1.s_value = 10;
WMP_publish(&t1);
```

Eliminar suscripción

```
Topic t1;
WMP_removeListener(&t1);
```