FREEMAES

Guía de Implementación

Daniel Andrés Rojas Marín

Versión 1.0.0

Tabla de contenidos

1. Visión General

FreeMAES corresponde a una biblioteca que implementa el paradigma de software MAES (Multi-Agent Framework for Embedded Systems) para el sistema operativo FreeRTOS. Opera como un complemento al sistema operativo para desarrollar aplicaciones con arquitectura basada en agentes.

Para realizar aplicaciones mediante la biblioteca FreeMAES primeramente hay que comprender los conceptos relacionados con:

- Sistemas multiagente
- El modelo FIPA para software multiagente
- El API de FreeRTOS

En esta guía si omitirán las explicaciones de componentes ajenos al contenido de la biblioteca.

Modelo de Clases para FreeMAES

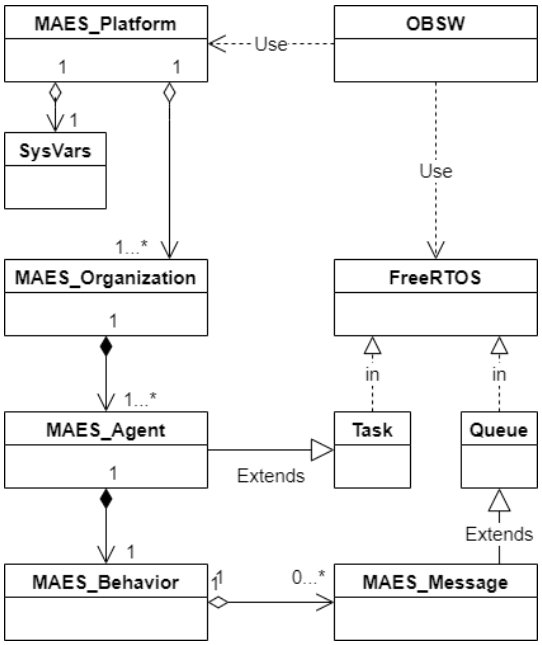


Figura 1. Modelo de clases para FreeMAES

Mediante las clases y relaciones mostradas en la figura 1, se puede notar que los agentes corresponden a contenedores de tareas mediante su comportamiento, que a su vez pueden incluir un sistema de mensajería que encapsula una cola dedicada por agente. Los agentes pueden asociarse por organizaciones designadas como equipos o jerarquías que pertenecen a una única plataforma de software contenida en el equipo electrónico. La clase sysVars funciona como un arreglo de variables de entorno que soporta otras funciones del paradigma.

2. API FreeMAES

A continuación, se desarrollan brevemente el propósito y capacidades de las clases que componen en API de FreeMAES. El desglose completo de métodos y estructuras de datos se encuentra al final del documento.

Agentes

Los agentes se crean mediante el método constructor de la clase `Agent`. El constructor de la clase solicita parámetros como el nombre del agente, la prioridad y la profundidad del stack (tal como se define en las tareas de FreeRTOS). Incluye un único método que retorna el identificador del agente (AID).

Comportamientos

Los comportamientos encapsulan las funciones de FreeRTOS que crean las tareas y buzones para los agentes. En este API se incluyen tres comportamientos con métodos virtuales: una clase genérica y dos subclases derivadas; comportamiento cíclico y *One Shot* (una sola ejecución). La principal diferencia entre las clases derivadas corresponde al método `done`, pues para el comportamiento cíclico siempre es verdadero y para el *One Shot* es falso.

El método `execute` asocia los métodos de cada clase y subclase como se muestra en la figura 2. Dicho método debe encerrarse con una función *wrapper* para ser asignado como la función de tarea de un agente.

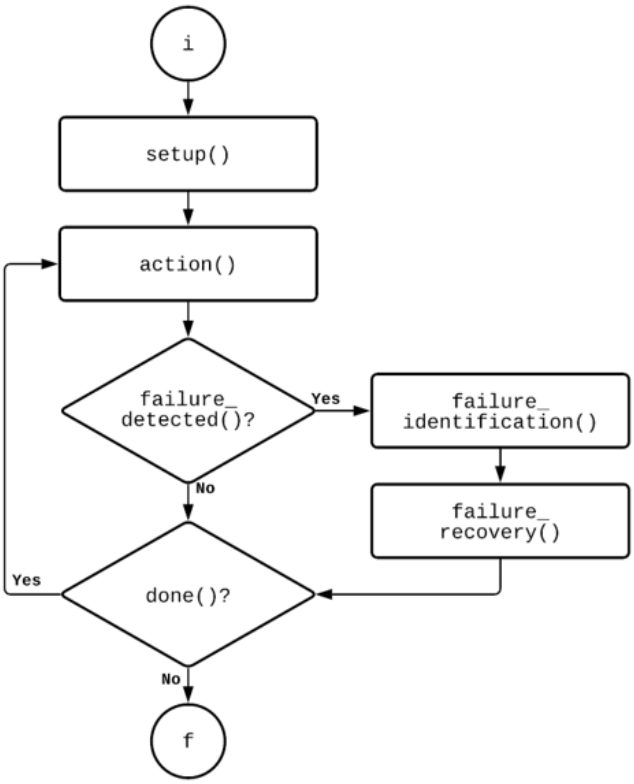


Figura 2. Diagrama de flujo del método Execute

Mensajes

Los mensajes transportan la información entre agentes; son etiquetados con tipo de mensaje, destinatario y remitente. El usuario directamente no debe crear cada mensaje pues están contenidos dentro de los comportamientos de cada agente. Cada agente posee un buzón (cola de FreeRTOS) con longitud unitaria. Para administrar la información de los mensajes el usuario debe usar los métodos de la clase `Agent_Message`.

Organizaciones

El constructor de la clase `Agent_Organization` requiere definir el tipo de organización: TEAM (equipo) o HEIRARCHY (jerarquía). En un equipo la comunicación está limitada a los miembros del equipo sin demás restricciones mientras que en una jerarquía los miembros solo pueden comunicarse con el moderador de su organización. La definición del dueño se ejecuta dentro del comportamiento del agente promocionado como dueño. Dicho agente puede enviar solicitudes mediante mensajes a otros agentes para unirse a la organización.

Plataforma

La clase `Agent_Platform` administra el registro e inicialización de los agentes, así como métodos misceláneos para el control de tareas. Los métodos de la clase son todos públicos, aunque están condicionados al agente (o mejor dicho tarea) que les convoque. Los métodos se pueden visualizar como funciones pre-calendarizador, funciones del Agente AMS y funciones no restringidas. Las funciones pre-calendarizador inician los agentes y arrancan el registro de los objetos a las variables de entorno de la plataforma, para estas funcionar es necesario que ninguna tarea se esté ejecutando. Las funciones del Agente AMS corresponden a registros, dadas de baja, terminaciones, suspensión y reanudación de los agentes. Finalmente, las funciones no restringidas producen demoras en las tareas (*Task Delays*), buscan agentes, descripciones y estados de agentes; entre otros.

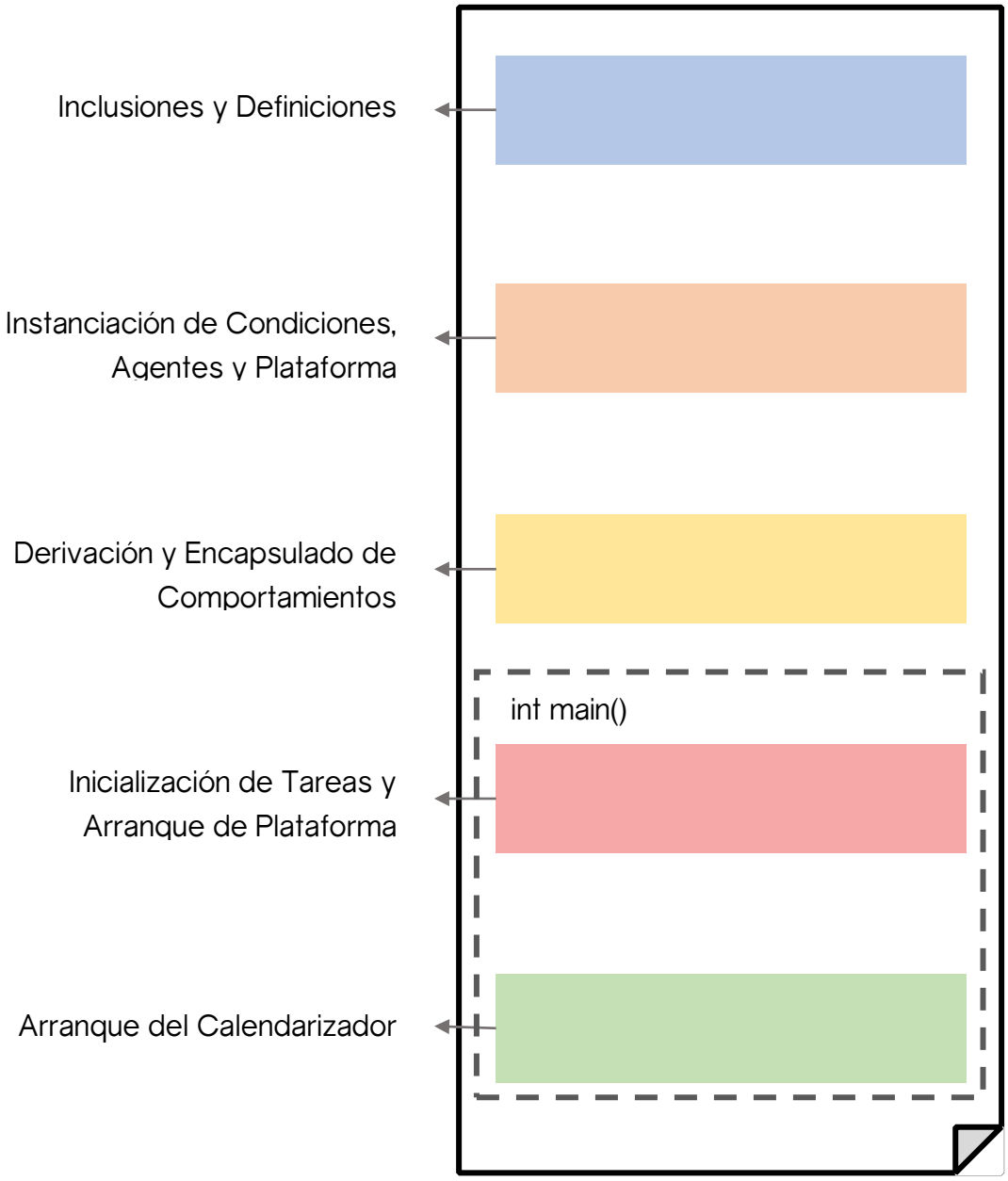
El constructor de esta clase solo solicita un parámetro; el nombre para la plataforma, sin embargo, puede iniciarse además con un parámetro más que define las condiciones para las funciones del Agente AMS mediante la clase `USER_DEF_COND`.

sysVars

Contendedor de variables de entorno, no debe ser usado por el usuario.

3. Estructura Básica de un Programa

El orden que se presenta a continuación corresponde a una recomendación general para organizar el desarrollo de un programa mediante el API FreeMAES. Se debe prestar mucha atención a las dependencias en los comportamientos hacia otros agentes o hacia la plataforma. El `main()` del programa debe finalizar con el arranque del calendarizador.



Inclusiones y Definiciones

En esta etapa inicial como en cualquier programa en C/C++ se incluyen las librerías a utilizar en la solución. Para incluir la API de FreeMAES, se debe incluir el archivo de cabecera "maes-rtos.h". La API está contenida dentro de un *namespace* de C++ llamado MAES, por lo que todas las declaraciones dentro de la API se acceden adjuntando "MAES::" frente a la función o variable global (como los enums).

Instanciación de Condiciones, Agentes y Plataformas

La instanciación de objetos se realiza invocando el constructor de cada clase a emplear. Los parámetros de cada constructor se describen en la sección 4. Se deben instanciar todos los Agentes a usar, las organizaciones que los contienen y una sola plataforma por equipo electrónico (microcontrolador o microcomputadora). Además, se pueden definir condiciones generales para las funciones del AMS e incluirlas en el constructor de plataforma.

Derivación y Encapsulado de Comportamientos

A partir de las clases `Generic_Behaviour`, `OneShotBehaviour` y `CyclicBehaviour` se deben describir la función de tarea como una subclase derivada. El método `setup()` define ajustes previos a un lazo de ejecución, el método `action()` describe la función principal de la tarea, los métodos `failure_detection()`, `failure_identification()` y `failure_recovery()` corresponden a los métodos para detección, identificación y recuperación de fallas respectivamente, y por último el método `done()` revisa si la función principal debe o no volver a ejecutarse. El orden de ejecución está coordinado por el método `execute()` cuyo diagrama se describe en la sección 2. Dentro de este flujo se debe definir a que agente se envían que mensajes y cuando esperar por mensajes (mediante la variable `msg` y los métodos de la clase `Agent_Msg`). Para ser incluidos en la tarea, el método `execute()` de cada subclase definida debe contenerse dentro de una función encapsuladora con el siguiente formato (incluyendo el caso de usar parámetros):

```
void   FuncionEncap(void* parametros){
       ComportamientoAgente b;
       b.taskParameters = parametros;
       b.execute();
       }
```

Inicialización de Tareas y Arranque de Plataforma

Dentro del `main()` de la solución, se debe inicializar cada agente, con o sin parametros, antes de arrancar el kernel. Mediante este método de plataforma se asigna una tarea a cada agente .Al terminar la inicialización se debe arrancar la plataforma con el método `boot()`.

Arranque de Calendarizador

La última función dentro del programa debe ser el arranque del calendarizador de FreeRTOS: `vStartScheduler()`.

4. Desglose de Clases del API FreeMAES

Clase Agent

Constructor de Agente: Crea la instancia de la clase. Requiere los parámetros de nombre, prioridad y tamaño de stack.

`Agent_AID AID()`: Entrega el AID del agente.

Clase Agent Message

Constructor de Mensaje: La instancia se crea dentro de la función encapsuladora.

`Mailbox_Handle get_mailbox(Agent_AID aid)`: Retorna el Mailbox Handle de la cola del agente según su AID.

`ERROR_CODE add_receiver(Agent_AID aid_receiver)`: Agrega el agente a la lista de receptores del agente que contiene el mensaje.

`ERROR_CODE remove_receiver(Agent_AID aid_receiver)`: Elimina al agente de la lista de receptores del agente que contiene el mensaje.

`void clear_all_receiver()`: Vacía la lista de receptores.

`void refresh_list()`: Recorre la lista y remueve receptores que ya no están registrados en la misma organización.

`bool isRegistered(Agent_AID aid)`: Revisa si el agente está registrado en la misma organización.

`void set_msg_type(MSG_TYPE type)`: Establece el tipo del mensaje.

`void set_msg_content(char* body)`: Establece el contenido del mensaje.

`MsgObj* get_msg()`: Recupera el registro del mensaje.

`MSG_TYPE get_msg_type()`: Recupera el tipo del mensaje.

`Char* get_msg_content()`: Establece el contenido del mensaje.

`Agent_AID get_sender()`: Recupera el AID del emisor del mensaje.

`Agent_AID get_target_agent()`: Recupera el AID del receptor del mensaje.

`MSG_TYPE receive(TickType_t timeout)`: Espera recibir un mensaje y bloquea la tarea por el periodo de timeut definido.

`ERROR_CODE send(Agent_AID aid_receiver, TickType_t timeout)`: Agrega un mensaje a la cola del agente objetivo en el periodo de timeout definido.

`ERROR_CODE send()`: Envía mensajes a todos los receptores de la lista.

`ERROR_CODE registration(Agent_AID target_agent)`: Envía una solicitud de registro al agente AMS para el agente objetivo.

`ERROR_CODE deregistration(Agent_AID target_agent)`: Envía una solicitud de registro al agente AMS para el agente objetivo.

`ERROR_CODE suspend(Agent_AID target_agent)`: Envía una solicitud de suspensión al agente AMS para el agente objetivo.

`ERROR_CODE resume(Agent_AID target_agent)`: Envía una solicitud de reanudación al agente AMS para el agente objetivo.

`ERROR_CODE kill(Agent_AID target_agent)`: Envía una solicitud de terminación al agente AMS para el agente objetivo.

`ERROR_CODE restart()`: Envía una solicitud de reinicio al agente AMS para el agente emisor.

Clase Agent Organization

Constructor de Organización: Crea la instancia organización. Requiere el tipo de organización.

`ERROR_CODE create()` : Se puede llamar desde un comportamiento. asigna a la organización su dueño (agente que llama la función) y la información del agente. La variable org apunta a la instancia de organización.

`ERROR_CODE destroy()`: Vacía las listas de la organización y las variables de rol, afiliación y org de cada miembro.

`ERROR_CODE isMember(Agent_AID aid)`: Revisa si el agente es miembro de la organización.

`ERROR_CODE isBanned(Agent_AID aid)`: Revisa si el agente está prohibido de la organización.

`ERROR_CODE change_owner(Agent_AID aid)`: Reasigna la posesión de la organización. Solo puede ser llamado por el dueño de la organización.

`ERROR_CODE set_admin(Agent_AID aid)`: Asigna la afiliación de administrador al agente. Solo puede ser llamado por el dueño de la organización.

`ERROR_CODE set_moderator(Agent_AID aid)`: Asigna el rol de moderador al agente. Solo puede ser llamado por el dueño de la organización.

`ERROR_CODE add_agent(Agent_AID aid)`: Agrega al agente a la organización y cambia su variable org a la organización de quien llama. Solo puede ser llamado por el dueño o el administrador de la organización.

`ERROR_CODE kick_agent(Agent_AID aid)`: Elimina al agente de la organización y cambia su variable org a NULL. Solo puede ser llamado por el dueño o el administrador de la organización.

`ERROR_CODE ban_agent(Agent_AID aid)`: Prohíbe al agente de la organización. Asigna el rol de moderador al agente. Solo puede ser llamado por el dueño o el administrador de la organización.

`ERROR_CODE remove_ban(Agent_AID aid)`: Elimina la prohibición en la organización sobre el agente. Solo puede ser llamado por el dueño o el administrador de la organización.

`void clear_ban_list()`: Limpia la lista de prohibiciones de la organización.

`ERROR_CODE set_participant(Agent_AID aid)`: Agrega al agente a la conversación y cambia su variable org a la organización de quien llama. Solo puede ser llamado por el dueño o el moderador de la organización.

`ERROR_CODE set_visitor(Agent_AID aid)`: Agrega al agente a la conversación como oyente y cambia su variable org a la organización de quien llama. Solo puede ser llamado por el dueño o el moderador de la organización.

`ORG_TYPE get_org_type()`: Regresa el tipo de organización.

`org_info get_info()`: Regresa la información de la organización.

`UBaseType_t get_size()`: Regresa el número de miembros de la organización.

`MSG_TYPE invite(Agent_Msg msg, UBaseType_t password, Agent_AID target_agent, UBaseType_t timeout)`: Envía una propuesta a otro agente para unirse a la organización.

Clase Agent Platform

Constructor de Plataforma: Crea la instancia de la clase. Requiere el nombre de la plataforma.

`bool boot()`: El método inicia la plataforma. Solo se puede llamar desde main().

`void agent_init(Agent* agent, void behaviour(void* pvParameters))`: El método asigna el comportamiento encapsulado en la función a la tarea y construye la cola del agente. Inicia la tarea con prioridad 0 y suspende su ejecución. Requiere la función y un puntero al agente.

`bool agent_search(Agent_AID aid)`: El método busca al agente según su AID y regresa verdadero si lo encuentra.

`void agent_wait(TickType_t ticks)`: llama a la función vTaskDelay de FreeRTOS y bloquea al agente por el tiempo definido.

`void agent_yield()`: El agente libera el procesador.

`Agent_AID get_running_agent()`: Regresa el AID del agente que actualmente se ejecuta.

`AGENT_STATE get_state(Agent_AID aid):` Regresa el estado del agente según su AID.

`Agent_info get_Agent_description(Agent_AID aid):` Regresa el registro con información del agente según su AID.

`AP_Description get_AP_description():` Regresa la descripción de la plataforma.

`ERROR_CODE register_agent(Agent_AID aid):` Función exclusiva del agente AMS. Asigna la prioridad adecuada a la tarea del agente y la activa. Incluye al agente en la plataforma.

`ERROR_CODE deregister_agent(Agent_AID aid):` Función exclusiva del agente AMS. Asigna la prioridad 0 a la tarea del agente y la desactiva. Remueve al agente de la plataforma.

`ERROR_CODE kill_agent(Agent_AID aid)`: Función exclusiva del agente AMS. Elimina la tarea.

`ERROR_CODE suspend_agent(Agent_AID aid)`: Función exclusiva del agente AMS. Suspende al agente.

`ERROR_CODE resume_agent(Agent_AID aid)`: Función exclusiva del agente AMS. Reanuda al agente.

`void restart(Agent_AID aid)`: Función exclusiva del agente AMS. Elimina la tarea del agente y la cola para luego restaurarla con diferente AID, mismos parámetros.

Clase sysVars

`void set_TaskEnv(Agent_AID aid, Agent* agent_ptr)`: Agrega el par de AID y puntero de agente a las variables de entorno.

`Agent* get_TaskEnv(Agent_AID aid)`: Regresa el puntero al agente que contiene el AID.

`void erase_TaskEnv(Agent_AID aid)`: Borra el par de AID y puntero de agente de las variables de entorno.

`sysVar* getEnv()`: Regresa el arreglo que contiene las variables de entorno.