ESPECIFICACIÓN DE AGENTES IP MÓVILES PARA PROCESOS AAA EN AMBIENTES 3G (ANEXOS)

ALEXANDER GALVIS QUINTERO

LUIS ALEJANDRO FLETSCHER BOCANEGRA

UNIVERSIDAD DEL CAUCA

FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES

DEPARTAMENTO DE TRANSMISIÓN

GRUPO NUEVAS TECNOLOGÍAS EN TELECOMUNICACIONES

POPAYÁN

2001

ESPECIFICACIÓN DE AGENTES IP MÓVILES PARA PROCESOS AAA EN AMBIENTES 3G (ANEXOS)

ALEXANDER GALVIS QUINTERO
LUIS ALEJANDRO FLETSCHER BOCANEGRA

Monografía presentada como requisito parcial para optar al título de Ingeniero en Electrónica y Telecomunicaciones.

DIRECTOR: Ing. OSCAR J. CALDERÓN C.

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
DEPARTAMENTO DE TRANSMISIÓN
GRUPO NUEVAS TECNOLOGÍAS EN TELECOMUNICACIONES
POPAYÁN
2001

TABLA DE CONTENIDO

TABLA DE FIGURAS

ANEXO A
PSEUDOCÓDIGOS PARA IMPLEMENTACIÓN DE CLASES DE DISEÑO

En este anexo se presentan los pseudocódigos que describen la manera como se debe hacer la implementación de las clases que conforman los agentes modelados.

> **Nota:** El pseudocódigo se realizó en un nivel alto de abstracción, por lo cual en algunos casos no es muy detallado y la forma exacta de implementar las operaciones y estructuras de datos dependen del lenguaje en el cual se escriban los agentes o se genere el código de los mismos.

A1. Implementación del Foreign Agent (FA)

A1.1. CLASE <<Entity>> Status

```
class Status
{
//Definición de los atributos y estructuras de datos
        private integer Id_Agente;
        private integer Id_Servidor;
        private integer Id_Movil;
        private array[1..8]integer Ubicacion;
        private array[1..10]string AS;
        private array[1..8]integer Dominio;

//Definición de los métodos e interfaces
        public Guardar_Status (mov, serv, agent, integer; ubi, dom
                array[1..8]integer);
        private Cambiar_Estado (est integer);
        private Cambiar_Ubicacion (ubi array[1..8]integer);
```

```
//Definición de las operaciones
        void Guardar_Status(mov, serv, agent, integer; ubi, dom
        array[1..8]integer)
        {
                Id_Agente:=agent;
                Id_Servidor:=serv;
                Id_Movil:=mov;
                Ubicacion:=ubi;
                Dominio:=dom;
        }

        void Cambiar_Estado(est integer)
        {
        Planeador.Estado:=est;
        }

        void Cambiar_Ubicacion(ubi array[1..8]integer)
        {
                Ubicacion:=ubi;
        }
}
```

A1.2. CLASE <<Entity>> Marco_Autenticacion

```
class Marco_Autenticación
{
// Definición de los atributos y estructuras de datos
        private hex Type;
        private hex Subtype;
        private hex Length;
        private hex SPI;
        private hex Chall_Auth;

//Definición de los métodos e interfaces
        public Verificar_Campos();
        public Guardar_Marco();
        public Establecer_Campos(ty, su, le, sp, c_a hex);
        public Leer_Campos();

//Definición de las operaciones
        integer Verificar_Campos()
        {
                Leer_Campos()
                Comprobar_longitud_de_campos();
                Comprobar_tipos_de_datos_de_los_campos();
```

```
            Verificar_presencia_total_de_los_campos();
            If (error) { return(0); }
            Else { return(1) }
        }

        void Guardar_Marco()
        {
            marco array[1..5]hex;

            marco[0]:=Type;
            marco[1]:=Subtype;
            marco[2]:=Length;
            marco[3]:=SPI;
            marco[4]:=Chall_Auth;
        }

        void Establecer_Campos(ty, su, le, sp, c_a hex)
        {
            Type:=ty;
            Subtype:=su;
            Length:=le;
            SPI:=sp;
            Chall_Auth:=c_a;
        }

        hex Leer_Campos(n integer)
        {
            Revisar_el_valor_contenido_en_el_atributo_indicado_por_n();
        }
}
```

A1.3. CLASE <<Entity>> Credencial_MN

```
class Credencial_MN
{
// Definición de los atributos y estructuras de datos
        private integer Id_Movil;
        private string Dominio;
        private array[1..8]hex Dir_IP;
        private array[1..8]hex CoA;

//Definición de los métodos e interfaces
        public Solicitar_Credenciales();
        public Guardar_Credenciales();
```

```
//Definición de las operaciones
        boolean Solicitar_Credenciales(cred array[1..3]string)
        {
                Id_Movil:=Exigir_Id();
                Dominio:=Exigir_Dom();
                Dir_IP:=Exigir_IP();
                Comparar_Credencial_con_datos_entregados();
                If (ERROR!) { return false; }
                Else { return true }
        }

        record Guardar_Credenciales()
        {
                record credencial
                {
                        mov integer;
                        dom string;
                        ip, coa array[1..8]hex;
                }

                credencial/mov:=Id_Movil;
                credencial/dom:=Dominio;
                credencial/ip:=Dir_IP;
                credencial/coa:=CoA;
                return(credencial);
                Anfitrion.Vincular_MN(credencial);
        }
}
```

A1.4. CLASE <<Control>> Piloto

```
class Piloto
{
// Definición de los atributos y estructuras de datos
        private array[1..8]hex Destino;
        private boolean Tipo_Trama;

//Definición de los métodos e interfaces
        private Suspender_Ejecucion();
        private Seleccionar_Destino();
        private Guardar_Status();
        private Seriacion();
        private Diseriacion();
        private Reanudar_Ejecucion();
        private Viajar();
```

```
        public Comprobar_Tipo_Trama();

//Definición de las operaciones

        void Suspender_Ejecucion()
        {
                stop_all_process();
        }

        array Seleccionar_Destino()
        {
                array[1..8]hex dest;

                if (Tipo_Trama=true) { Destino:=Dirección del Servidor AAA; }
                if (Tipo_Trama=false) { Destino:=Dirección del Móvil; }
        }

        void Guardar_Status()
        {
                integer mov1, serv1, agent1;
                array[1..8]integer ubi1, dom1;

                mov1:=Consulta_mov();
                serv1:=Consulta_serv();
                agent1:=Consulta_agent;
                ubi1:=Consulta_ubi;
                dom1:=Consulta_dom;
                Status.Guardar_Status(mov1, serv1, agent1, ubi1, dom1);
        }

        void Seriacion()
        {
                Empaquetar_Status_Agente();
                Empaquetar_Codigo_Agente();
                Empaquetar_Marco();
                Convertir_paquetes_a_serie_binaria();
        }

        void Diseriacion()
        {
                Convertir_serie_binaria_a_paquetes();
                Desempaquetar_Status_Agente();
                Desempaquetar_Codigo_Agente();
                Desempaquetar_Marco();
        }
```

```
        void Reanudar_Ejecucion();
        {
                start_all_process();
        }

        void Viajar()
        {
                Establecer_conexion_con_destino();
                Transmitir_serie_binaria_del_agente();
                Autoeliminarse();
        }

        boolean Comprobar_Tipo_Trama()
        {
                boolean tipo;
                hex dato;

                dato:=Marco_Autenticacion.Leer_Campos(1);
                If (dato= "Requerimiento de autenticación") { tipo:=true };
                If (dato= "Respuesta de autenticación") { tipo:=false };
                return(tipo);
        }
}
```

A1.5. CLASE <<Control>> Planeador

```
class Planeador
{
// Definición de los atributos y estructuras de datos
        private boolean Resultado;
        public integer Estado;
        private array[1..8]integer Ubicacion;

//Definición de los métodos e interfaces
        public Verificar();
        private Leer_Resultado();
        private Actualizar_Status();
        public Recibir_Marco();
        public Reconocer_Destino();
        private Entregar_Marco();

//Definición de las operaciones
        void verificar()
        {
```

```
        Resultado:=Marco_Autenticacion.Verificar_Campos();
}

boolean Leer_Resultado()
{
        return(Resultado);
}

void Actualizar_Status()
{
        integer mov1, serv1, agent1;
        array[1..8]integer ubi1, dom1;

        mov1:=Consulta_mov();
        serv1:=Consulta_serv();
        agent1:=Consulta_agent;
        ubi1:=Consulta_ubi;
        dom1:=Consulta_dom;
        Status.Guardar_Status(mov1, serv1, agent1, ubi1, dom1);
}

void Recibir_Marco(trama array[1..5]hex)
{
        Marco_Autenticación.Establecer_Campos(trama[0], trama[1],
                trama[2],trama[3],trama[4]);
}

void Reconocer_Destino(dest boolean)
{
        string dir, dominio, usuario;
        boolean res;

        dir:=Consultar_Direccion;
        dominio:=Consultar_Dominio;
        usuario:=Consultar_Usuario;
        if (dir→dominio)
        {
                if (usuario=null) { res:=true};
                else { res:=false };
        }
        if (res=dest) { Entregar_Marco() };
        else { ERROR! };
}

void Entregar_Marco()
```

```
        {
                array[1..5]hex trama1
                integer i=0;

                for (i=0, i++, i=4)
                {
                        trama1[i]:=Marco_Autenticacio.Leer_Campos[i];
                }
                Interfaz.Enviar_Mensaje(trama1);
        }
}
```

A1.6. CLASE <<Control>> Anfitrion

```
class Anfitrion
{
// Definición de los atributos y estructuras de datos
        private boolean Resultado_Búsqueda;
        private array[1..4]string Mensajes;

//Definición de los métodos e interfaces
        private Actualizar_Status();
        private Buscar_Movil(info1 string);
        public Presentarse_a_MN();
        private Comprobar_MN(info2 array[1..3]string);
        public Recibir_Info_Movil(info array[1..3]string);
        public Vincular_MN(record cred2 { mov integer; dom string; ip, coa
                                array[1..8]hex; });

//Definición de las operaciones
        void Actualizar_Status()
        {
                integer mov1, serv1, agent1;
                array[1..8]integer ubi1, dom1;

                mov1:=Consulta_mov();
                serv1:=Consulta_serv();
                agent1:=Consulta_agent;
                ubi1:=Consulta_ubi;
                dom1:=Consulta_dom;
                Status.Guardar_Status(mov1, serv1, agent1, ubi1, dom1);
        }

        void Buscar_Movil(info1 array[1..3]string)
        {
```

```
            string idm;
            record cred1
            {
                mov integer;
                dom string;
                ip, coa array[1..8]hex;
            }

            Resultado_Busqueda:=false;
            while (Resultado_Busqueda=false)
            {
                Viajar_a_movil()
                idm:=Consultar_ID_del_movil();
                if (idm=info1[0]) { Comprobar_MN(info1) };
                else { Viajar_a_otro_movil() };
            }
            cred1:=Credencial_MN.Guardar_Credenciales();
        }

        void Presentarse_a_MN()
        {
            Pasar_ID_Agente_al MN();
            Asignar_CoA_al_MN();
        }

        void Comprobar_MN(info2 array[1..3]string)
        {
            Resultado_Busqueda:=
                Credencial_MN.Solicitar_Credenciales(info2);

        }

        void Recibir_Info_Movil(info array[1..3]string)
        {
            Buscar_Movil(info);
        }

        void Vincular_MN(record cred2 { mov integer; dom string;
                ip, coa array[1..8]hex; })
        {
            Pasar_datos_del_MN_al_AAAFS();
            Presentarse_a_MN();
        }
}
```

A1.7. CLASE <<Boundary>> Interfaz

```
class Interfaz
{
// Definición de los atributos y estructuras de datos
        private integer Tipo;
        private integer Estado();
        private array[..]data Mensaje;
        private array[1..8]string Destino;

//Definición de los métodos e interfaces
        public Enviar_Mensaje(msg1 array[..]data);
        public Recibir_Mensaje(msg2 array[..]data);

//Definición de las operaciones
        void Enviar_Mensaje(msg1 array[..]data);
        {
                Determinar_tipo_de_mensaje_a_enviar();
                Mensaje:=Adaptar_mensaje(msg1);
                Transmitir_mensaje(Destino, Mensaje);
        }

        void Recibir_Mensaje(msg2 array[..]data);
        {
                Determinar_tipo_de_mensaje_a_RECIBIR();
                Recpcion_De_mensaje();
                Mensaje:=Adaptar_mensaje(msg2);
        }
}
```

A2. Implementación del Intermediary Agent (IA)

A2.1. CLASE <<Entity>> Status

```
class Status
{
//Definición de los atributos y estructuras de datos
        private integer Id_Agente;
        private integer Id_Servidor;
        private integer Id_Movil;
        private array[1..8]integer Ubicacion;
        private array[1..10]string AS;
        private array[1..8]integer Dominio;

//Definición de los métodos e interfaces
        public Guardar_Status (mov, serv, agent, integer; ubi, dom
                array[1..8]integer);
        private Cambiar_Estado (est integer);
        private Cambiar_Ubicacion (ubi array[1..8]integer);

//Definición de las operaciones
        void Guardar_Status(mov, serv, agent, integer; ubi, dom
        array[1..8]integer)
        {
                Id_Agente:=agent;
                Id_Servidor:=serv;
                Id_Movil:=mov;
                Ubicacion:=ubi;
                Dominio:=dom;
        }

        void Cambiar_Estado(est integer)
        {
        Planeador.Estado:=est;
        }

        void Cambiar_Ubicacion(ubi array[1..8]integer)
        {
                Ubicacion:=ubi;
        }
}
```

A2.2. CLASE <<Entity>> Marco_Autenticacion

```
class Marco_Autenticación
{
// Definición de los atributos y estructuras de datos
        private hex Type;
        private hex Subtype;
        private hex Length;
        private hex SPI;
        private hex Chall_Auth;

//Definición de los métodos e interfaces
        public Verificar_Campos();
        public Guardar_Marco();
        public Establecer_Campos(ty, su, le, sp, c_a hex);
        public Leer_Campos();

//Definición de las operaciones
        integer Verificar_Campos()
        {
                Leer_Campos()
                Comprobar_longitud_de_campos();
                Comprobar_tipos_de_datos_de_los_campos();
                Verificar_presencia_total_de_los_campos();
                If (error) { return(0); }
                Else { return(1) }
        }

        void Guardar_Marco()
        {
                marco array[1..5]hex;

                marco[0]:=Type;
                marco[1]:=Subtype;
                marco[2]:=Length;
                marco[3]:=SPI;
                marco[4]:=Chall_Auth;
        }

        void Establecer_Campos(ty, su, le, sp, c_a hex)
        {
                Type:=ty;
                Subtype:=su;
                Length:=le;
                SPI:=sp;
```

```
            Chall_Auth:=c_a;
        }

        hex Leer_Campos(n integer)
        {
            Revisar_el_valor_contenido_en_el_atributo_indicado_por_n();
        }
}
```

A2.3. CLASE <<Entity>> SA_Status

```
class SA_Status
{
// Definición de los atributos y estructuras de datos
        private array[1..10]string SA_Home;
        private array[1..10]string SA_Foreign;
        private boolean SAH_Status;
        private boolean SAF_Status;

//Definición de los métodos e interfaces
        private Verificar_SA(i boolean);
        public Fijar_SA(j boolean;  s array[1..10]string);

//Definición de las operaciones
        boolean Verificar_SA(i boolean)
        {
            est boolean;

            if (i=true) { est:=SAH_Status };
            else { est:=SAF_Status };
            return(est);
        }

        void Fijar_SA(j boolean;  s array[1..10]string);
        {
            est1, res boolean;

            res:=Comparar_parámetros_de_asociación_
                        de_seguridad(j boolean);
            if (res=true)
            {
                if (j=true) { SA_Home:=s; SAH_Status:=true };
                else { SA_Foreign:=s; SAF_Status:=true }
            };
            else (ERROR!);
```

```
            est1:=Verificar_SA(j);
            if (est1≠true) { ERROR! };
        }
}
```

A2.4. CLASE <<Entity>> Info_Dominio

```
class Info_Dominio
{
// Definición de los atributos y estructuras de datos
        private string DominioH;
        private string DominioF;
        private integer Id_AAAFS;
        private integer Id_AAAHS;

//Definición de los métodos e interfaces
        public Verificar_Dominio(i boolean);
        public Verificar_Servidor(j boolean);
        public Establecer_Campos();

//Definición de las operaciones
        boolean Verificar_Dominio(i boolean);
        {
            dom string;
            ver1 boolean;

            if (i=true)
            {
                dom:=Consultar_dominio_H();
                if (dom=Dominio_H) { ver1:=true) };
                else { ver1:=false };
            }
            else
            {
                dom:=Consultar_dominio_F();
                if (dom=Dominio_F) { ver1:=true) };
                else { ver1:=false };
            }
            return(ver1);
        }

        boolean Verificar_Servidor(j boolean);
        {
            serv string;
            ver2 boolean;
```

```
                if (j=true)
                {
                        serv:=Consultar_servidor_H();
                        if (serv=Id_AAASH) { ver2:=true) };
                        else { ver2:=false };
                }
                else
                {
                        serv:=Consultar_servidor_F();
                        if (dom=Id_AAASF) { ver2:=true) };
                        else { ver2:=false };
                }
                return(ver2);
        }

        void Establecer_Campos();
        {
                DominioH:=Consultar_dominio_H();
                DominioF:=Consultar_dominio_F();
                Id_AAAFS:=Preguntar_idH();
                Id_AAAHS:=Preguntar_idF();
        }

}
```

A2.5. CLASE <<Control>> Planeador

```
class Planeador
{
// Definición de los atributos y estructuras de datos
        private boolean Resultado;
        public integer Estado;
        private array[1..8]integer Ubicacion;

//Definición de los métodos e interfaces
        public Verificar();
        private Leer_Resultado();
        private Actualizar_Status();
        public Recibir_Marco();
        public Reconocer_Destino();
        private Entregar_Marco();

//Definición de las operaciones
        void verificar()
```

```
{
        Resultado:=Marco_Autenticacion.Verificar_Campos();
}

boolean Leer_Resultado()
{
        return(Resultado);
}

void Actualizar_Status()
{
        integer mov1, serv1, agent1;
        array[1..8]integer ubi1, dom1;

        mov1:=Consulta_mov();
        serv1:=Consulta_serv();
        agent1:=Consulta_agent;
        ubi1:=Consulta_ubi;
        dom1:=Consulta_dom;
        Status.Guardar_Status(mov1, serv1, agent1, ubi1, dom1);
}

void Recibir_Marco(trama array[1..5]hex)
{
        Marco_Autenticación.Establecer_Campos(trama[0], trama[1],
                trama[2],trama[3],trama[4]);
}

void Reconocer_Destino(dest boolean)
{
        string dir, dominio, serv1, serv2;
        boolean res;

        dir:=Consultar_Direccion;
        dominio:=Consultar_Dominio;
        usuario:=Consultar_Usuario;
        if (dir→dominio)
        {
                if (serv1=null) { res:=true};
                else { res:=false };
        }
        if (res=dest) { Entregar_Marco() };
        else { ERROR! };
}
```

```
        void Entregar_Marco()
        {
                array[1..5]hex trama1
                integer i=0;

                for (i=0, i++, i=4)
                {
                        trama1[i]:=Marco_Autenticacio.Leer_Campos[i];
                }
                Interfaz.Enviar_Mensaje(trama1);
        }
}
```

A2.6. CLASE <<Control>> Piloto

```
class Piloto
{
// Definición de los atributos y estructuras de datos
        private array[1..8]hex Destino;
        private boolean Tipo_Trama;

//Definición de los métodos e interfaces
        private Suspender_Ejecucion();
        private Seleccionar_Destino();
        private Guardar_Status();
        private Seriacion();
        private Diseriacion();
        private Reanudar_Ejecucion();
        private Viajar();
        public Comprobar_Tipo_Trama();

//Definición de las operaciones

        void Suspender_Ejecucion()
        {
                stop_all_process();
        }

        array Seleccionar_Destino()
        {
                array[1..8]hex dest;

                if (Tipo_Trama=true) { Destino:=Dirección del Servidor AAAH; }
                if (Tipo_Trama=false) { Destino:=Dirección del Servidor AAAF; }
        }
```

```
void Guardar_Status()
{
        integer mov1, serv1, agent1;
        array[1..8]integer ubi1, dom1;

        mov1:=Consulta_mov();
        serv1:=Consulta_serv();
        agent1:=Consulta_agent;
        ubi1:=Consulta_ubi;
        dom1:=Consulta_dom;
        Status.Guardar_Status(mov1, serv1, agent1, ubi1, dom1);
}

void Seriacion()
{
        Empaquetar_Status_Agente();
        Empaquetar_Codigo_Agente();
        Empaquetar_Marco();
        Convertir_paquetes_a_serie_binaria();
}

void Diseriacion()
{
        Convertir_serie_binaria_a_paquetes();
        Desempaquetar_Status_Agente();
        Desempaquetar_Codigo_Agente();
        Desempaquetar_Marco();
}

void Reanudar_Ejecucion();
{
        start_all_process();
}

void Viajar()
{
        Establecer_conexion_con_destino();
        Transmitir_serie_binaria_del_agente();
        Autoeliminarse();
}

boolean Comprobar_Tipo_Trama()
{
        boolean tipo;
```

```
                hex dato;

                dato:=Marco_Autenticacion.Leer_Campos(1);
                If (dato= "Requerimiento de autenticación") { tipo:=true };
                If (dato= "Respuesta de autenticación") { tipo:=false };
                return(tipo);
        }
}
```

A2.7. CLASE <<Control>> Intermediario

```
class Intermediario
{
// Definición de los atributos y estructuras de datos
        private boolean Estado_Asociacion;
        private array[1..4]string Mensajes;

//Definición de los métodos e interfaces
        private Verificar_Estado_Asociacion(j boolean);
        private Establecer_Asociacion();
        private Actualizar_Status();
        private Presentarse_a_AAAS(i boolean);

//Definición de las operaciones
        boolean Verificar_Estado_Asociacion(j boolean);
        {
                est boolean;

                if (j=true) { est:=SAH_Status };
                else { est:=SAF_Status };
                return(est);
        }

        void Establecer_Asociacion();
        {
                boolean res1, res2;

                Determinar_dominio_y_servidor_foraneo();
                Conectar_con_servidor_foraneo();
                res1:=Solicitar_verificacion_de_usuario();
                If (res1=true) { Presentarse_a_AAAS(res1);
                Else { ERROR! };
                Determinar_dominio_y_servidor_local();
                Conectar_con_servidor_local();
                res2:=Solicitar_verificacion_de_usuario();
```

```
            if (res2=true) { Presentarse_a_AAAS(res2);
            else { ERROR! };
        }

        void Actualizar_Status();
        {
            integer mov1, serv1, agent1;
            array[1..8]integer ubi1, dom1;

            mov1:=Consulta_mov();
            serv1:=Consulta_serv();
            agent1:=Consulta_agent;
            ubi1:=Consulta_ubi;
            dom1:=Consulta_dom;
            Status.Guardar_Status(mov1, serv1, agent1, ubi1, dom1);
        }

        void Presentarse_a_AAAS(i boolean);
        {
            if (i=true) { Pasar_ID_agente_al_AAAFS() };
            else { Pasar_ID_agente_al_AAAHS() };
        }
}
```

A2.8. CLASE <<Boundary>> Interfaz

```
class Interfaz
{
// Definición de los atributos y estructuras de datos
        private integer Tipo;
        private integer Estado();
        private array[..]data Mensaje;
        private array[1..8]string Destino;

//Definición de los métodos e interfaces
        public Enviar_Mensaje(msg1 array[..]data);
        public Recibir_Mensaje(msg2 array[..]data);

//Definición de las operaciones
        void Enviar_Mensaje(msg1 array[..]data);
        {
            Determinar_tipo_de_mensaje_a_enviar();
            Mensaje:=Adaptar_mensaje(msg1);
            Transmitir_mensaje(Destino, Mensaje);
        }
```

```
        void Recibir_Mensaje(msg2 array[..]data);
        {
                Determinar_tipo_de_mensaje_a_RECIBIR();
                Recpcion_De_mensaje();
                Mensaje:=Adaptar_mensaje(msg2);
        }
}
```

ANEXO B
DIAGRAMAS DE COLABORACIÓN

En este anexo se presentan los diagramas de colaboración elaborados para los casos de uso en los cuales se basó el modelado del sistema.

B1. Casos de uso para el Foreign Agent (FA)

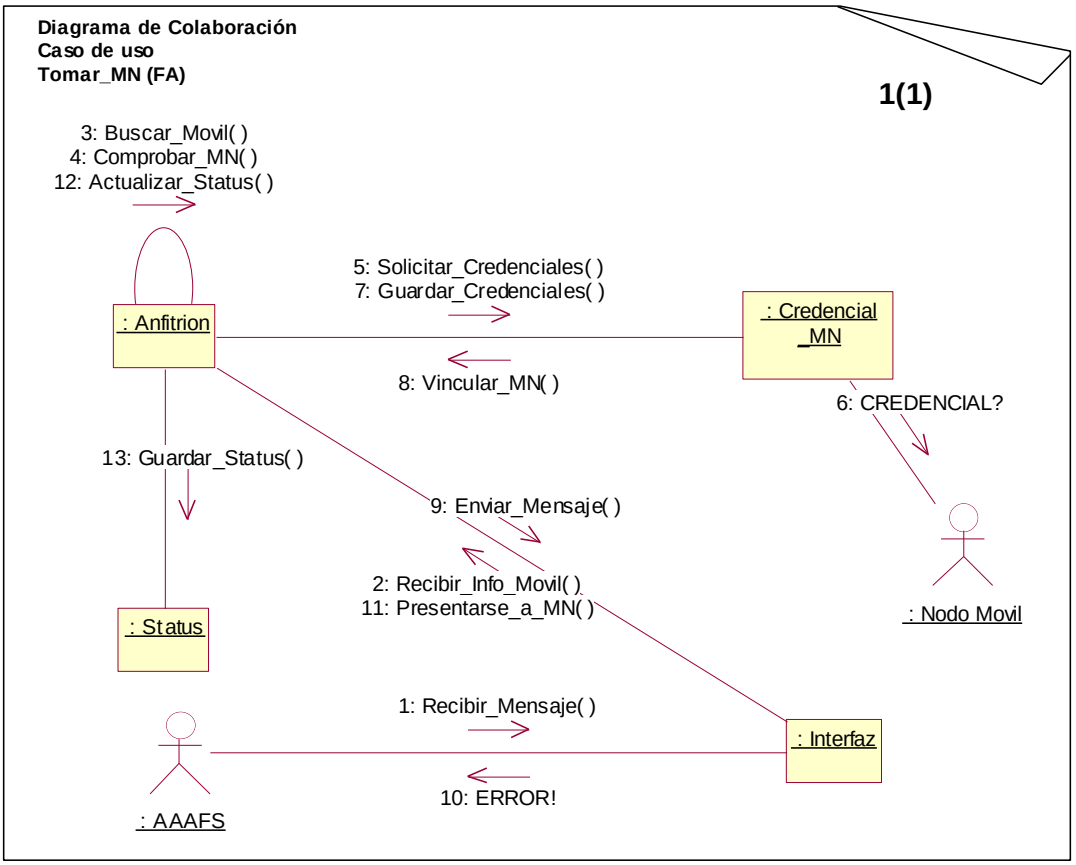

Figura 1 Caso de Uso Tomar_MN

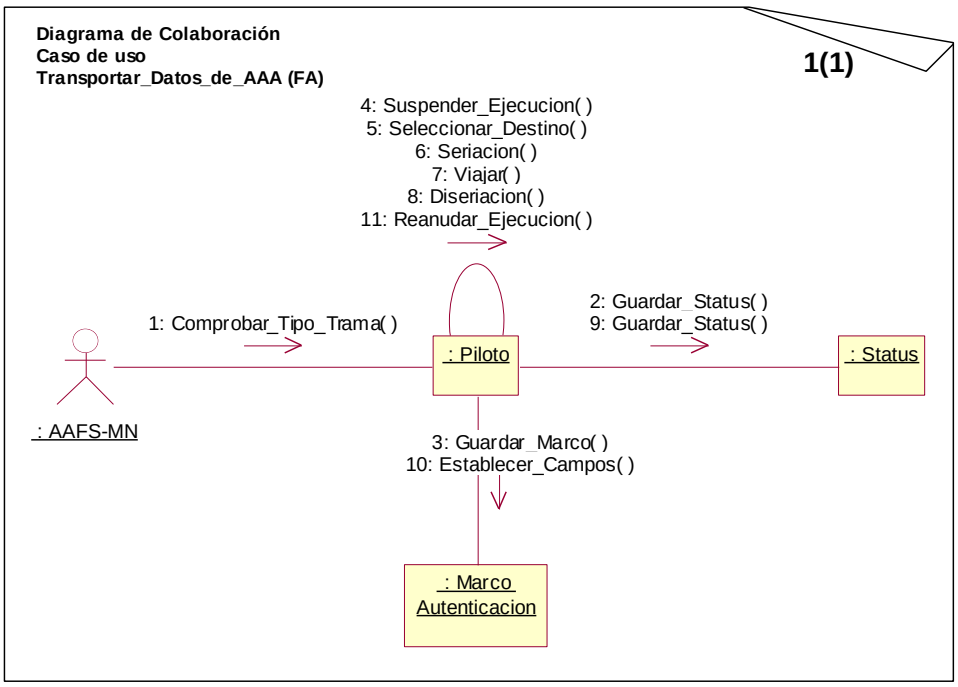

Figura 2 Caso de Uso Atender_Solicitud_de_Registro

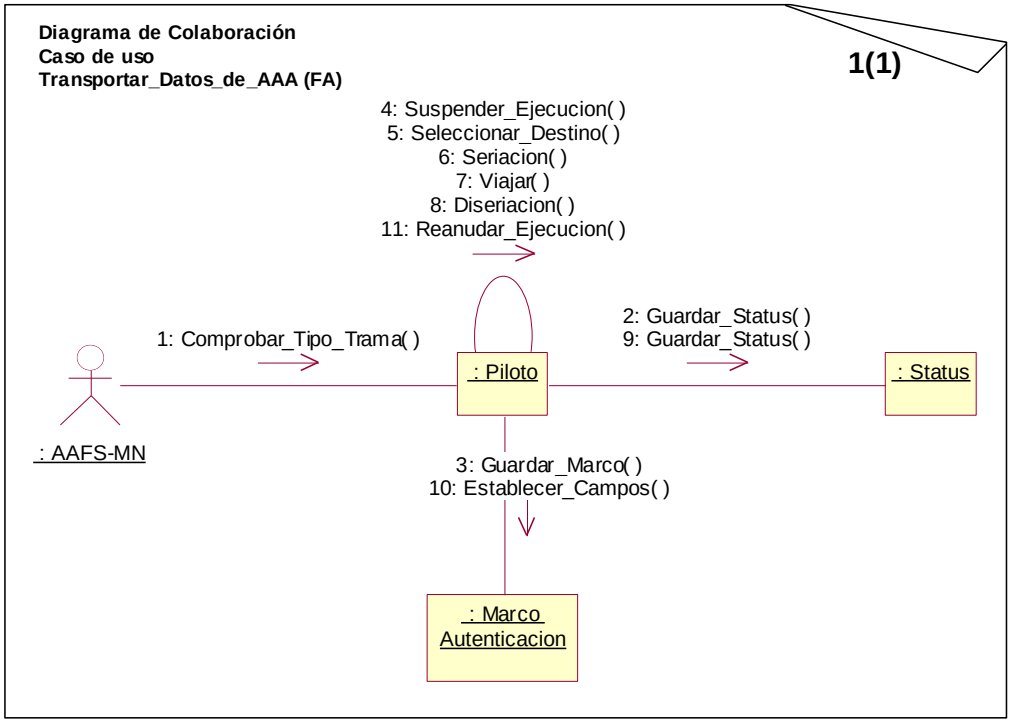

Figura 3 Caso de Uso Transportar_Datos_de_AAA

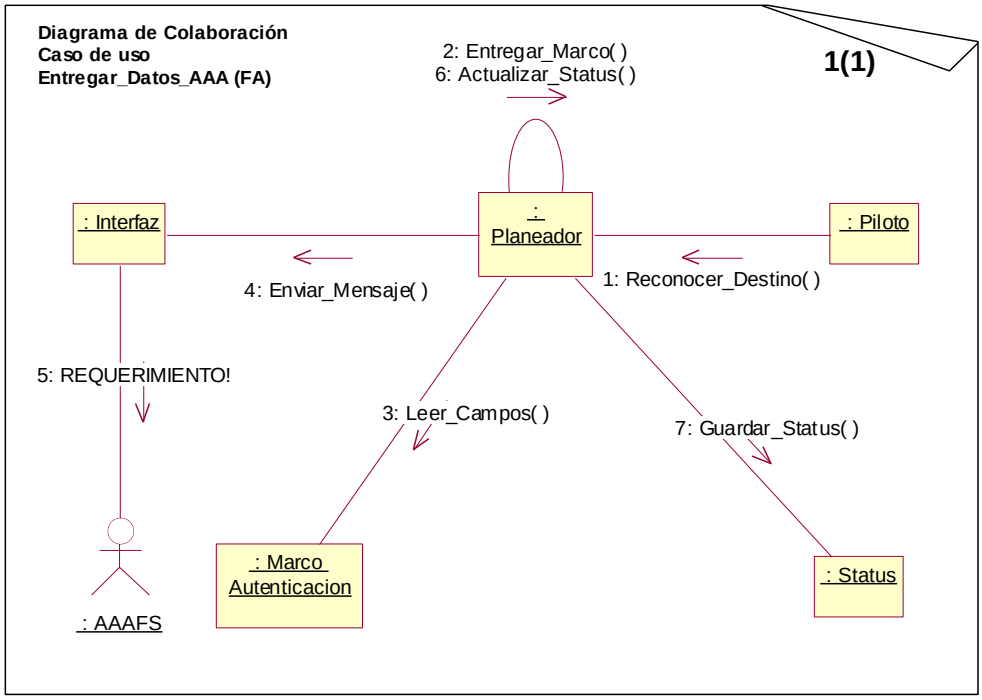

Figura 4 Caso de Uso Entregar_Datos_de_AAA

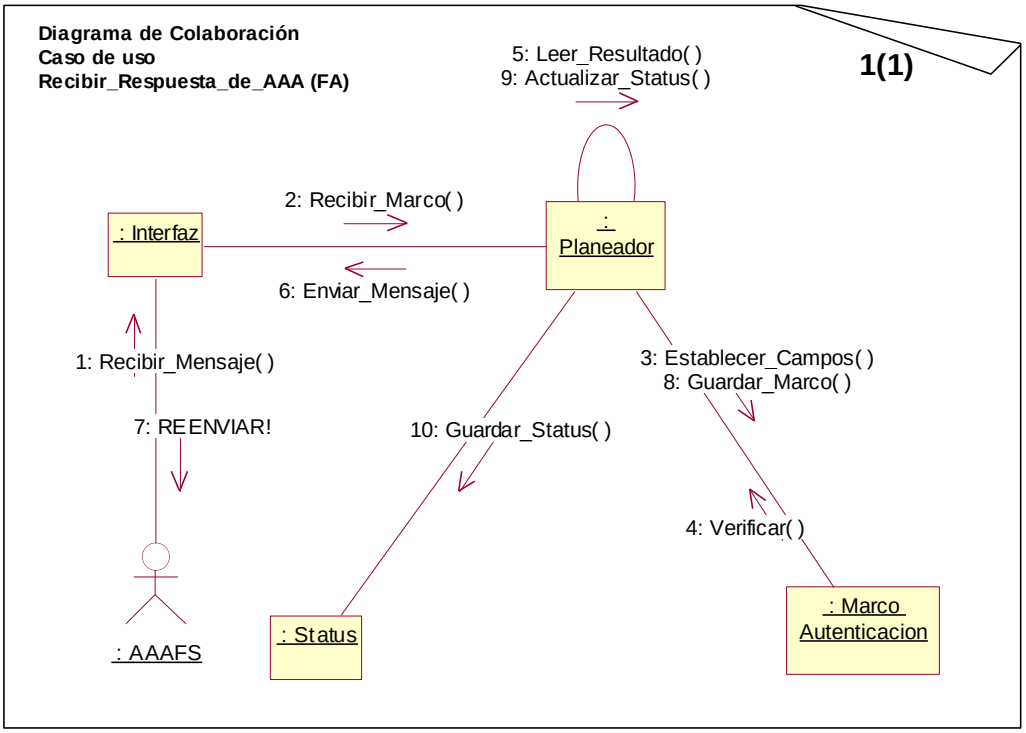

Figura 5 Caso de Uso Recibir_Respuesta_de_AAA

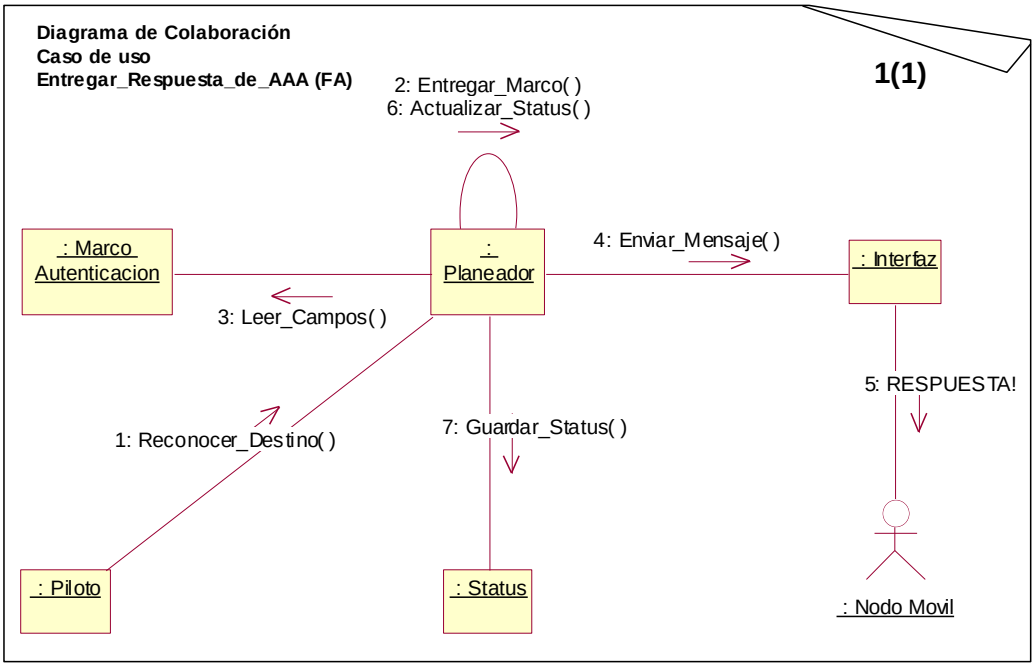

Figura 6 Caso de Uso Entregar_Respuesta_de_AAA

B2. Casos de uso para el Intermediary Agent (IA)

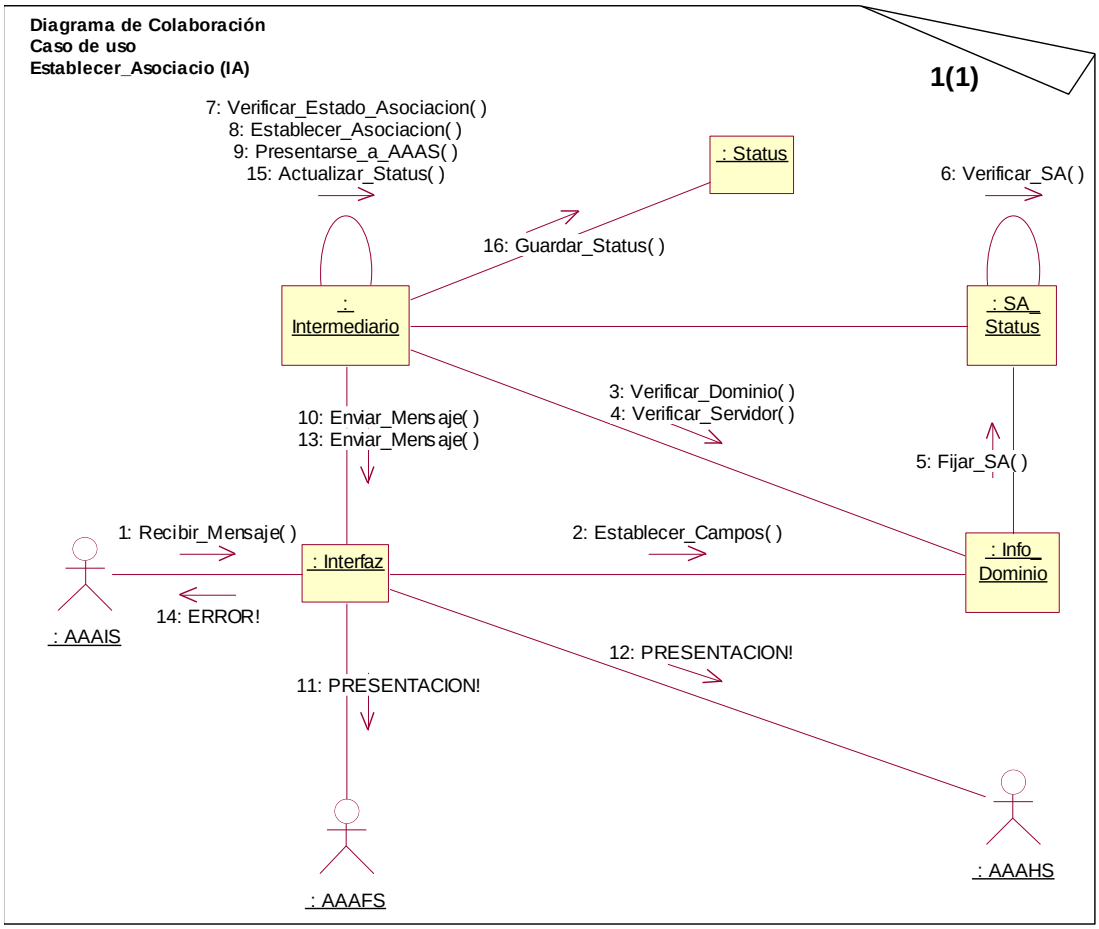

Figura 7 Caso de Uso Establecer_Asociacion

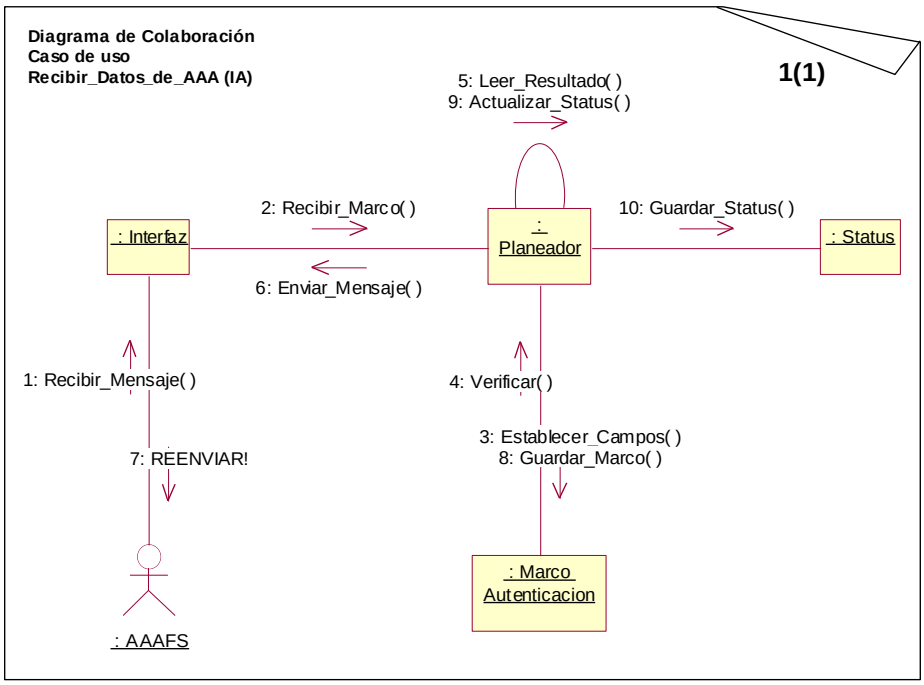

Figura 8 Caso de Uso Recibir_Datos_de_AAA

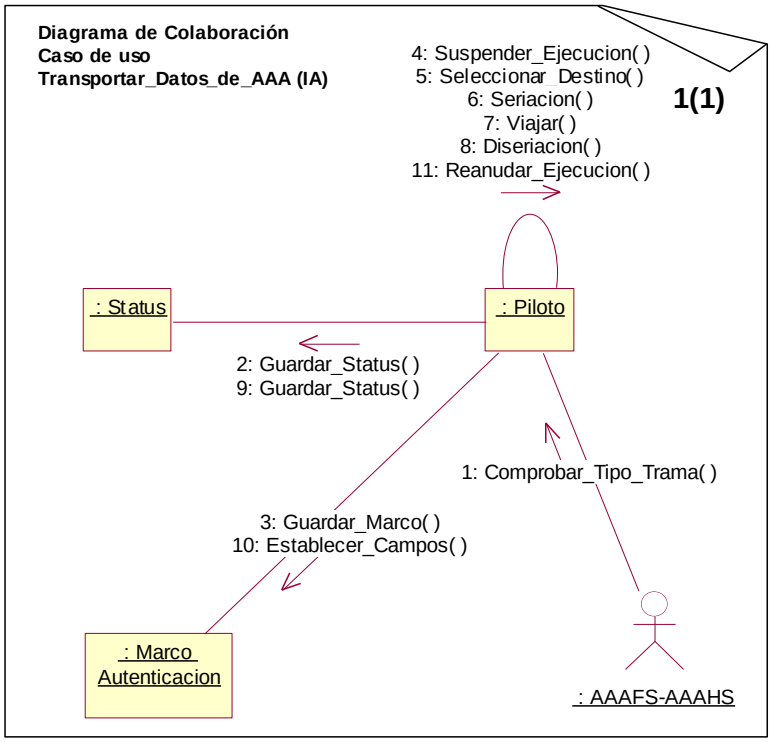

Figura 9 Caso de Uso Transportar_Datos_de_AAA

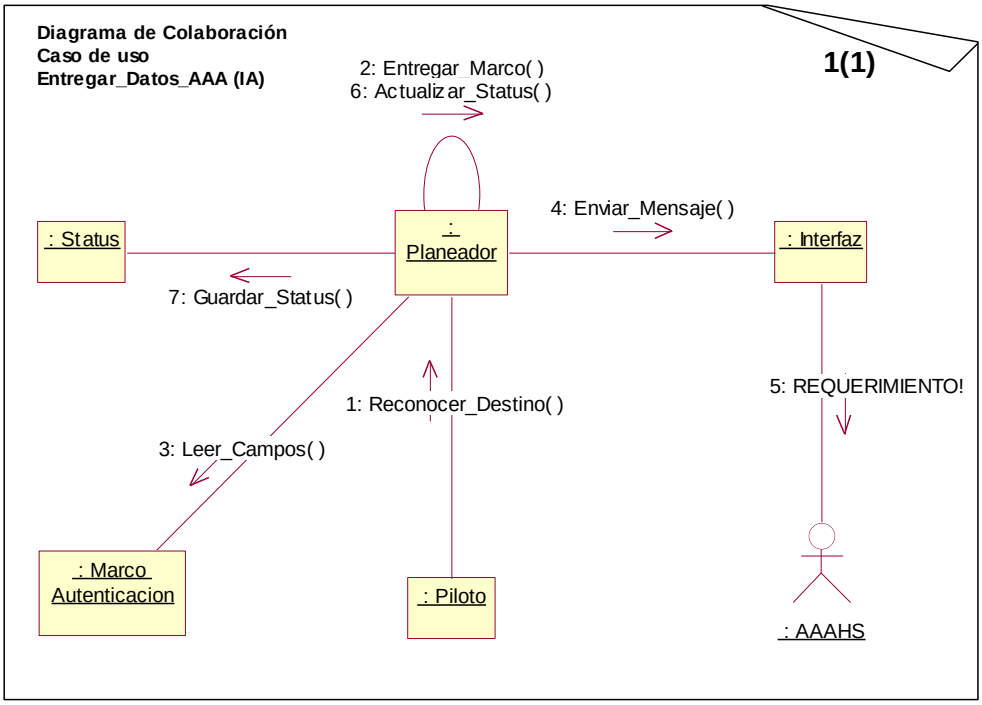

Figura 10 Caso de Uso Entregar_Datos_de_AAA

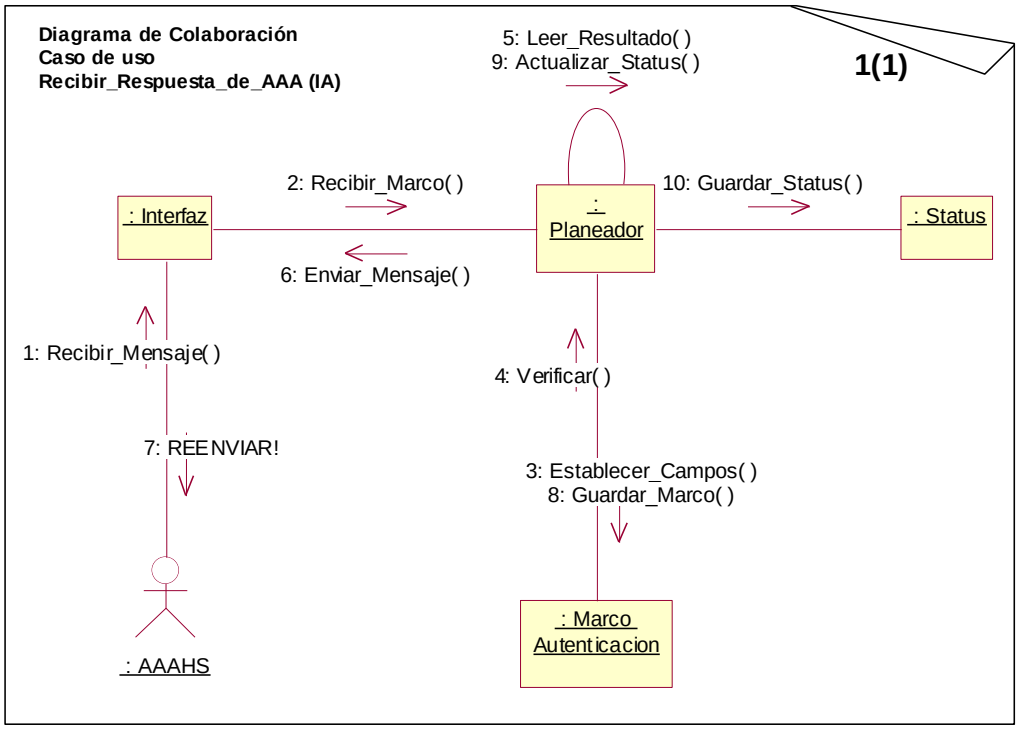

Figura 11 Caso de Uso Recibir_Respuesta_de_AAA

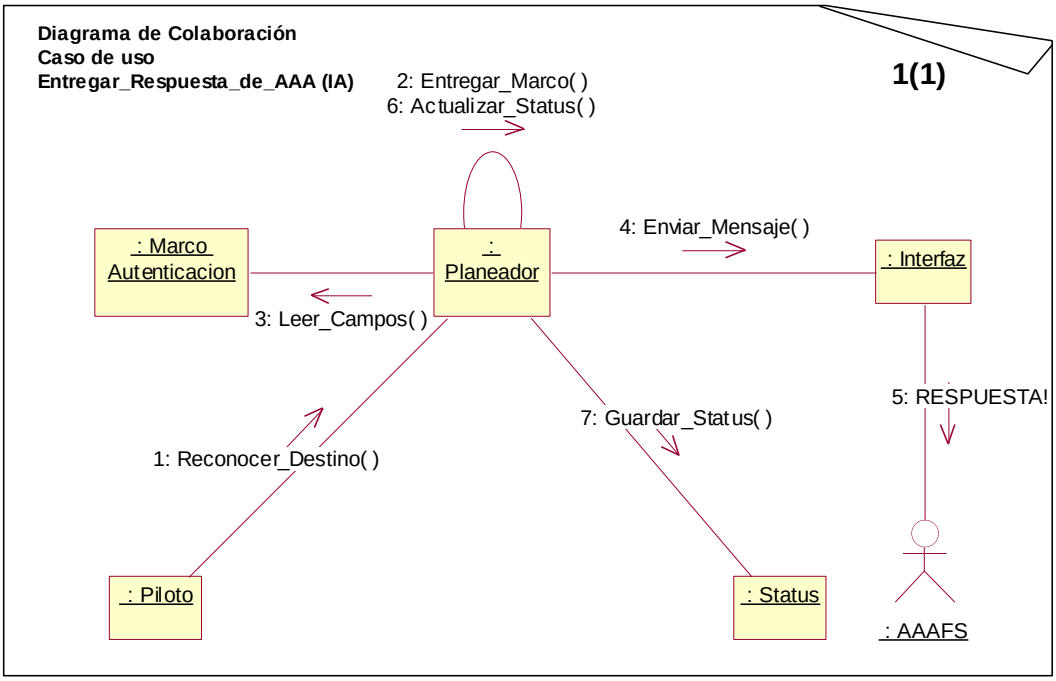

Figura 12 Caso de Uso Entregar_Respuesta_de_AAA

ANEXO C
DICCIONARIO DEL SISTEMA

En este anexo se realizan descripciones y definiciones de términos y conceptos utilizados en el modelado de los agentes (Capítulo 4 de la Monografía), y que son relevantes para la correcta comprensión y asimilación del funcionamiento del sistema.

AAA (Authentication, Authorization, Accounting): Es la abreviatura para los tres procesos básicos de seguridad y gestión de usuarios en la red y los servicios que se le prestan.

Agente Foráneo (FA: Foreign Agent): Agente encargado de mediar la provisión de servicio a un móvil que se encuentra en una red que no es la propia (roaming).

Agente Intermediario (IA: Intermediary Agent): Agente encargado de mediar el intercambio de información entre dos servidores y dominios que no comparten una asociación de seguridad.

Asociación de Seguridad (SA: Security Association): Conjunto de información que comparten dos dominios sobre los registros VLR y HLR. Se aclara que esta información es compartida pero nunca transportada por la red radio.

Autenticación: Proceso mediante el cual se comprueba la identidad alegada por un dispositivo o usuario.

Care of Address: Dirección IP perteneciente al dominio foráneo y que es asignada a través del agente foráneo al móvil que se encuentra en roaming para que opere dentro de la red mientras está en dicha condición.

CoA: Abreviatura de Care of Address.

Chall_Auth: Variable que contiene el valor del último campo del Marco de Autenticación o Respuesta de Autenticación. En el primer caso contiene el Challenge generado por el móvil, y en el segundo caso contiene el Authenticator entregado por los servidores AAA.

Challenge: Es un valor (número) generado pseudoaleatoriamente por el móvil a partir de la información del mismo (ESN, Id_Móvil, entre otros datos).

Destino: Variable que contiene la dirección de destino hacia la cual viaja el agente móvil.

Dir_IP: Variable que contiene la dirección IP (IPv6 address) perteneciente al móvil.

Diseriación: Proceso mediante el cual el Sistema de Gestión de Agentes convierte el código recibido en una estructura que conforman el agente viajero y la información que transporta.

Dominio Foráneo (DominioF): Dominio en el cual se encuentra un nodo móvil solicitando servicios y que es diferente a su dominio propio.

Dominio Local (DominioH): Dominio al cual pertenece originalmente un móvil que actualmente se encuantra en roaming.

Dominio Mediador: Dominio que proporciona un agente intermediario.

Estado: Variable que contiene un valor que representa el estado en el cual se encuentra el agente.

Estado_Asociación: Variable que contiene un valor que indica si una asociación se encuentra o no establecida.

Id_AAAFS: Variable que contiene un identificador del servidor AAA del dominio foráneo.

Id_AAAHS: Variable que contiene un identificador del servidor AAA del dominio local.

Id_Agente: Variable que contiene el identificador del agente.

Id_Movil: Variable que contiene el identificador del nodo móvil.

Id_Servidor: Identificador de un servidor determinado.

Length: Variable que almacena un campo del marco de autenticación que detalla la longitud total del marco.

Marco: Es una trama perteneciente al protocolo AAA. En este caso posee cinco campos y puede ser un Requerimiento o una Respuesta de Autenticación.

Nodo Móvil (MN: Mobile Node): Entidad que representa al equipo de usuario.

Requerimiento de Autenticación: Tipo de marco que contiene la información generada por el MN y que posibilita al mismo solicitar un servicio a la red en la cual se encuentra.

Respuesta de Autenticación: Tipo de marco que contiene la información generada por el AAAS y que posibilita al mismo comunicar al MN la respuesta a su solicitud de servicio a la red en la cual se encuentra.

SA_Foreign: Variable que contiene parte de la información que permite establecer una SA con el dominio foráneo y verificar la validez y el estado de la misma.

SA_Home: Variable que contiene parte de la información que permite establecer una SA con el dominio local y verificar la validez y el estado de la misma.

SAF_Status: Variable que guarda el estado de la SA con el dominio foráneo, es decir, si está o no establecida correctamente.

SAH_Status: Variable que guarda el estado de la SA con el dominio local, es decir, si está o no establecida correctamente.

Seriación: Proceso mediante el cual el Sistema de Gestión de Agentes y el agente mismo convierte el código del agente, los datos de éste y la información que transporta en un código serial listo para ser transmitido.

Servidor AAA Foráneo (AAAFS: AAA Foreign Server): Servidor que administra el dominio foráneo y encargado de los procesos AAA en el mismo.

Servidor AAA Intermediario (AAAIS: AAA Intermediary Server): Servidor que administra el dominio mediador y encargado de proveer el agente intermediario.

Servidor AAA Local (AAAHS: AAA Home Server): Servidor que administra el dominio local y encargado de los procesos AAA en el mismo.

SPI: Campo que especifica los algoritmos de autenticación y criptografía, las claves para dichos algoritmos, el tiempo de vida de las mismas, los vectores de inicialización y la dirección de origen de la asociación.

Status: Clase que contiene la información que en conjunto definen el estado actual de un agente.

Subtype: Campo del marco de autenticación que contiene un valor que define el subtipo del mismo.

Tipo_Trama: Variable que contiene un valor que indica si el marco actual es un Requerimiento o una Respuesta de autenticación.

Type: Campo del marco de autenticación que contiene un valor que define el tipo del mismo.

Ubicación: Variable que contiene el dominio actual en el cual se encuentra el agente móvil.

Nota: Otros términos relacionados se encuentran definidos en el Glosario de la Monografía y a lo largo de todo el documento.