INSTITUTO TECNOLOGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS ESTADO DE MEXICO

IMPLEMENTACIÓN DE UN SISTEMA COMPUTACIONAL MULTIDOMINIO

TESIS QUE PRESENTA

SALVADOR CONTRERAS HERNÁNDEZ

MAESTRÍA EN CIENCIAS COMPUTACIONALES
MCCR 93

Diciembre, 1997

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY
CAMPUS ESTADO DE MÉXICO

IMPLEMENTACIÓN DE UN SISTEMA DE SEGURIDAD MULTIDOMINIO

T E S I S
QUE PARA OPTAR EL GRADO DE MAESTRO EN CIENCIAS COMPUTACIONALES
P R E S E N T A

SALVADOR CONTRERAS HERNÁNDEZ

Asesor: Dr. José de Jesús Vázquez Gómez

Comité de tesis: Dr. Luis Ángel Trejo Rodríguez
Dr. Roberto Gómez Cárdenas
Dr. José de Jesús Vázquez Gómez

Jurado:	Dr. Luis Ángel Trejo Rodríguez	Presidente
	Dr. Roberto Gómez Cárdenas	Secretario
	Dr. José de Jesús Vázquez Gómez	Vocal

Atizapán de Zaragoza, Edo. de Méx., Diciembre de 1997

ÍNDICE

	Página
1 Introducción	7
2 Control de acceso	10
3 Políticas de seguridad computacional	11
3.1 Política de seguridad Multinivel	11
3.2 Política de seguridad Comercial	13
3.2.1 Un modelo de integridad	13
3.3 Política de seguridad Financiera	16
3.3.1 Organización de los datos	16
3.3.2 Violaciones indirectas	18
4 Enfoques de seguridad Multidominio	19
4.1 El enfoque de ECMA	19
4.2 Multipolíticas	22
4.3 El enfoque de L.J. La Padula	25
4.4 Lógica de Seguridad	28
4.4.1 Concepto de fórmula bien formada	28
4.4.2 Permiso y Obligación	29
4.5 Álgebra de seguridad	32
4.6 Restricciones en el sistema de vistas del usuario	35
4.6.1 Restricción	38
4.6.1.1 Máquina de estado restrictiva	38
4.7 Requerimientos para la interacción entre Dominios	39
4.8 Tipos de interacciones entre Dominios	40
4.8.1 Consideraciones	41
5 Enfoque de control mandatorio	42
5.1 Descripción del enfoque	42
5.2 Nivel de abstracción	44
5.3 Jerarquía de propiedades	44
5.4 Determinación del nivel de abstracción	46
5.5 Seguridad y Vivacidad	46
5.6 Jerarquía del nivel de abstracción	47
5.7 Multipolítica de interacción	48
5.8 Valores mínimos de atributos	48
5.9 Condiciones de la Multipolítica	49
5.10 Nivel de interacción	50
5.11 Propiedades básicas	52
6 Implementación de la Autoridad Multidominio	53
6.1 Función de la Autoridad Multidominio	53
6.1.1 Forma de evaluación de la Autoridad Multidominio	59
6.2 Análisis de la Autoridad Multidominio	60
6.2.1 Acceso fuertemente restrictivo	61

7 Administración del sistema 64
7.1 Administración de usuarios 66
7.2 Administración de Dominios 70
7.3 Selección de puertos 78
7.4 Atributos de la información 81
7.5 Implementación de la comunicación 82
7.6 Políticas locales 87
7.7 Transferencia de mensajes 91
7.8 Jerarquía de propiedades 94
7.9 Generación de informes de auditoría 94
7.10 Trabajos futuros de investigación 97
8 Conclusiones 98
9 Referencias 100
ANEXO A. Reporte de auditoría 102
ANEXO B. Funciones de librería usadas en la comunicación 112
ANEXO C. Programas del sistema de Seguridad Multidominio 114

ÍNDICE DE FIGURAS

Figura	Página
4.1.1 Relación de subdominio	20
4.1.2 Relación par	20
4.2.1 Principales componentes de la máquina multipolítica	24
4.3.1 Política de control de acceso	26
4.6.1 Interconexión de A y B	36
4.8.1 Tipos de interacción	40
5.1.1 Dominios de seguridad	43
5.3.1 Elementos del multidominio	45
6.1.1 MLS, política fundamental	54
6.1.2 Flujo de datos en la autorización	55
6.1.3 Estructura de políticas por host	55
6.1.4 enviar obtiene valores de atributos y se transfieren a la AM	57
6.1.5 Datagrama utilizado en la interacción	58
6.2.1.1 MLS como interacción mínima	62
6.2.1.2 Interacción permitida para la información no sensible en D_o	62
7.1.1 Inserción de usuario nuevo	66
7.1.2 Consulta de usuarios	67
7.1.3 Modificaciones de atributos de usuario	69
7.2.1 Escritura de nuevo host con sus políticas	72
7.2.2 Consulta de políticas por host	73
7.2.3 Selección para cambiar atributos de host	75
7.2.4 Validación de argumentos recibidos por *hostcam*	76
7.2.5 Funciones de modificación para MLS	77
7.2.6 Lista de hosts	77
7.3.1 Conexión al puerto del host destino	78
7.3.2 Localización de un puerto	79
7.3.3 Código para ingresar un host con un puerto asociado	80
7.5.1 Fragmento del código de la AM	83
7.5.2 Código de AM para recepción de mensajes provenientes del cliente	84
7.5.3 Flujo de datos en el sistema	84
7.5.4 Envío de respuesta de AM al cliente	85
7.5.5 Código del hijo de AM	86
7.6.1 Orden de evaluación de políticas	87
7.6.2 Código de búsqueda de usuario	88
7.6.3 Valores de atributos de información y usuarios	89
7.6.4 Decisión para F	90
7.6.5 Envío de respuesta a la AM	90
7.7.1 Envío de mensajes	91

7.7.2 Parte del código del servidor de mensajes 92
7.7.3 Código del cliente para envío de mensajes 93
7.9.1 Creación del reporte de auditoría 95
7.9.2 Reporte de auditoría 96

ÍNDICE DE TABLAS

Tabla	Página
5.10.1 Ejemplo 1 de niveles de interacción	50
5.10.2 Ejemplo 2 de niveles de interacción	51
5.10.3 Ejemplo 3 de niveles de interacción	51
6.1.1 Políticas por host	56
7.1 Atributos de usuario	65
7.1.1 Listado de usuarios del sistema	70
7.3.1 Puertos asociados a cada host	78

1 INTRODUCCIÓN

En la actualidad es necesario implementar sistemas de seguridad que permitan mantener la confidencialidad e integridad de la información. Esto, en parte, es consecuencia de la conectividad que están sufriendo los sistemas de cómputo en general. En el futuro se podrá observar un número mayor de interacciones ya que existirán mecanismos que permitan transferencias de información seguras.
Para empezar a hablar de los enfoques de seguridad computacional es necesario definir lo siguiente:

* Objetos: Recursos a los cuales se puede tener acceso, tales como CPU, impresora, memoria, archivos, directorios, etc.
* Sujetos: Aquellos que pueden realizar el acceso a los objetos, tales como usuarios, procesos, etc. Nótese que un proceso inactivo puede ser considerado como un recurso, es decir como un objeto.
* Política de seguridad: Conjunto de reglas precisas para un sistema computacional, que permiten conservar la información íntegra, prevenir su divulgación no autorizada, así como contar con un control de autentificación que comprueba cuales son los objetos y sujetos del sistema.
* Dominio de seguridad: Un grupo de objetos y sujetos de seguridad a los cuales se les aplica una sola política de seguridad por un sólo administrador de seguridad.

La autorización depende de los atributos de seguridad (privilegios de los usuarios y de la información).

Los dominios pueden estar constituidos por una combinación de varias políticas. Esta combinación será una nueva política.
Actualmente existen varios problemas en el ámbito de la seguridad computacional. Estos problemas se refieren, entre otros, a que los sistemas no son flexibles, es decir, que si una parte de la política de seguridad es modificada, se tienen que evaluar los puntos sensibles nuevamente; además el intercambio de información en tiempo real entre políticas diferentes es muy difícil o imposible. También hay que tomar en cuenta que en la realidad existen múltiples políticas diferentes.
Cuando dos sistemas de seguridad de dominios diferentes desean interactuar de forma segura, es posible que no puedan lograr esto, debido a que los criterios de seguridad varían de un sistema a otro. Para solucionar este problema se proponen varios enfoques. Uno de ellos en particular, es en el que todas las interacciones entre las políticas puedan ser controladas por medio de un multipolítica, que se denominará también como Autoridad Multidominio (AM) [1]. Por lo tanto, las políticas del enfoque deben definirse dependiendo del entorno del multidominio.

El objetivo de este trabajo es determinar si es posible llevar a cabo una interacción eficiente entre dominios de seguridad diferentes, utilizando para esto el enfoque descrito en [1].

Debido a razones de confidencialidad, integridad y disponibilidad, estas interacciones deben garantizar la seguridad de acuerdo a las políticas establecidas.

Actualmente existen algunas propuestas sobre políticas de seguridad para tratar de que interactuen dominios diferentes. Sin embargo, no existen aún herramientas formales para especificar y analizar estas nuevas interacciones entre dominios. Este problema de interacción se denomina ***problema de seguridad multidominio***. Hay dos razones principales que explican este problema [1] :

1) Los dominios que tienen diferentes autoridades de seguridad: Los controles de seguridad varían de un sistema a otro ya que los criterios de seguridad seleccionados por las autoridades locales son diferentes.

2) Los dominios que tienen diferentes entornos: Las políticas de seguridad difieren ampliamente debido a las diferencias del entorno en el cual operan las organizaciones. La diversidad del entorno se presenta en niveles diferentes:

   - Diferentes Países: Cada nación propone sus propias normas y estándares para hacer respetar la seguridad nacional.

   - Diferentes sectores socioeconómicos: Cada sector tiene requerimientos de seguridad particulares, los cuales no son necesariamente compartidos por otros sectores (militar, comercial, salud, financiero, etc.).

   - Productos diferentes: Existen diferencias, aunque sean sutiles, entre los productos para la implementación de una política de seguridad, por lo cual puede ser que los productos no interactuen como se espera, aún si estos implementan la misma política de seguridad (Por ejemplo, Sistemas Operativos y Sistemas de Base de Datos).

Todas estas diferencias hacen difícil la comunicación segura entre dominios heterogéneos. Además de esto, las necesidades de interacción que involucran políticas de seguridad diferentes son cada vez más grandes, por lo cual el desarrollo de los productos que implementen la seguridad en los sistemas de cómputo tiene que llevarse a cabo conjuntamente con el desarrollo de productos que permitan dicha comunicación.

En la actualidad se han propuesto varias políticas de seguridad. Algunas de estas se desarrollan en sistemas distribuidos, ya que es común la interacción entre dominios de seguridad distintos en los sistemas distribuidos, por lo cual dichas interacciones deben garantizar la seguridad de acuerdo a las políticas establecidas.

Dentro de las políticas más estudiadas se encuentran: política Multinivel [4], Comercial [6] y Financiera [7]. Estas políticas son también las que se consideraron para desarrollar el presente trabajo y su implementación se encuentra en detalle en el trabajo de Angélica Ramírez [3].

En el resto de este trabajo se describen algunos enfoques sobre Seguridad Multidominio. Se estudia especialmente el enfoque propuesto en [1] ya que la implementación del sistema expuesto en esta tesis está basado en él. También se explica en detalle la función de la Autoridad

Multidominio, la cual es la que regula las interacciones entre los dominios, así como los programas para el cliente y el servidor en el dominio destino y los programas para la administración del sistema. Posteriormente se exponen los resultados que se obtuvieron al hacer interactuar las políticas descritas por Angélica Ramírez [3] bajo el control de la Multipolítica mencionada. También se presenta un panorama de los trabajos futuros para esta investigación.

2 CONTROL DE ACCESO

El control de acceso es un aspecto importante en cualquier sistema de seguridad. Uno de los modelos de control de acceso es el propuesto por la ISO [13]. En este modelo existe un *iniciador* y un *objetivo*, en donde el primero intenta accesar al segundo. El acceso puede ser permitido o negado por medio de una función que hace respetar el control de acceso AEF (Access Enforcer Function). El permitir o negar el acceso depende de una función de decisión ADF (Access Decission Function) . Para tomar esta decisión la ADF depende de la información de control de acceso ACI (Access Control Information). La ACI debe estar asociada al iniciador, al objetivo o a la información contextual. Por ejemplo, la identidad del iniciador es una clase típica de ACI. Los derechos de acceso para realizar operaciones como leer, escribir, borrar, etc., también pueden ser una clase de ACI.
Si un iniciador no es autentificado entonces el control descrito anteriormente no es muy eficaz porque podría ser sustituido por uno falso.
Un esquema de control de acceso es el denominado *lista de control de acceso* ACL (Access Control List), el cual consiste en ligar al objetivo una lista de identidades de iniciador y las acciones que éste puede ejecutar. Posteriormente, el ADF checa si la identidad del iniciador y la acción se encuentran en la lista. Esta característica se puede contemplar como elemento básico en la política Comercial descrita más adelante.

Otro esquema es el siguiente: Todas las ACI se ligan al iniciador, de esta manera, contienen la identidad del objetivo y permiten ciertas acciones. EL siguiente paso es el revisar si la identidad del objetivo en el ACI corresponde a la identidad real. A este esquema de acceso se le denomina *capacidades*.

3 POLÍTICAS DE SEGURIDAD COMPUTACIONAL

Este capítulo es una revisión de las tres políticas de seguridad computacional utilizadas en la implementación del Sistema de Seguridad Multidominio. Las políticas referidas son Seguridad Multinivel (Multilevel Security), Comercial y Financiera [3]. De ahora en adelante la referencia que se harán de estas políticas podrán ser con abreviaciones como MLS, C y F para la Multinivel, Comercial y Financiera respectivamente. La razón de la selección de estas políticas se debe a que son políticas ampliamente conocidas. Las políticas en los dominios dentro de la implementación del Sistema Multidominio son una combinación de MLS, C y F. Puede existir en un host[1] una, dos o tres políticas. Más adelante se explicará que la política MLS es considerada como política de base dentro del enfoque de J. Vázquez [1]. Aún con esta condición de manejar valores de base en el sistema, se puede incluir en la lista de Dominios algunas máquinas que no posean la política MLS. En caso de que un usuario quiera enviar información a algún host de estos, simplemente la comunicación será negada. El mismo caso sucede para un cliente que no posea las políticas propuestas para el desarrollo del sistema Multidominio, cuando la información que se transmite es confidencial.

3.1 Política de Seguridad Multinivel

La política Multinivel o, también conocida como de seguridad militar, es el conjunto de reglas que fueron propuestas para controlar la información clasificada del gobierno de Estados Unidos de América[10], con estas reglas se pretende que toda la información sea protegida contra su divulgación, es decir, que la información sea confidencial. Las reglas para la política Multinivel aparecen al el *Trusted Computer System Evaluation Criteria* [10], también conocido como *Orange Book.*

La seguridad de un sistema posee la propiedad de confidencialidad si, la información manipulada por éste no es disponible ni puesta en descubierto para usuarios, entidades o procesos no autorizados.

Las reglas de seguridad para los documentos del Departamento de Defensa de E.U., conocidas bajo el nombre de Política de Seguridad Multinivel o Política de Seguridad Militar especifican que:

- Toda información tiene asociada una clase de seguridad, es decir, que todo conjunto de datos está etiquetado según su clasificación.

- Toda persona tiene asociado un nivel de autorización, es decir, a cada usuario le es asignado un nivel que determina a qué datos puede tener acceso y a cuales no.

[1] máquina o nodo de la red de computadoras.

- La naturaleza de las clasificaciones de seguridad y de los niveles de autorización está conformada por dos partes :
  1) Un nivel de seguridad (No clasificado, Confidencial, Secreto y Super Secreto).
  2) Un conjunto de categorías (Seguridad, OTAN, ONU, Nuclear, etc.).

Por lo tanto una persona tiene derecho de **leer** un documento sí su nivel de autorización es superior o igual a la clasificación de la información y además, sí el conjunto de las categorías de la persona domina al conjunto de categorías de la información.

La razón por la cual se debe clasificar la información es para controlar que no sea divulgada a individuos no autorizados. Para resolver por completo el problema de versiones no autorizadas de información se requiere un mecanismo de control de escritura tan bueno como el de lectura. Esta política fue formalizada por Bell y LaPadula [4]. El mecanismo de control de lectura y escritura que propusieron considera dos propiedades:

- **Seguridad simple**: Condición de lectura hacia abajo. Un sujeto (usuario) tiene acceso de lectura a un objeto (información) sí la clasificación de seguridad del sujeto domina (es superior o igual) a aquella del objeto.

- **Propiedad *** : Condición de escritura hacia arriba. Un sujeto tiene acceso en escritura sobre un objeto sí la clasificación de seguridad del objeto domina a aquella del sujeto.

La clasificación de seguridad está compuesta de un nivel de seguridad y un conjunto de categorías. Un importante componente del mecanismo utilizado en esta política es que revisa la clasificación de todas las lecturas y escrituras, aunque estos controles no son suficientes para hacer cumplir las reglas de seguridad militar porque los usuarios autorizados no pueden ser totalmente confiables. Por lo tanto, para hacer cumplir esta política, el sistema debe protegerse a sí mismo, tanto de los usuarios no autorizados como de los autorizados, ya que éstos últimos pueden desclasificar información como resultado de un error, como una acción ilegal deliberada o porque invoquen un programa que sin su consentimiento desclasifique la información (caballos de troya). La desclasificación también puede ocurrir por el copiado de información.

Además, el Orange Book considera otros mecanismos que deben ser usados para poner en práctica la seguridad, podemos mencionar la autentificación y autorización de usuarios, generación de información de auditoría y asociación de etiquetas de control de acceso. Aunque el énfasis es la confidencialidad, esto no quiere decir que la integridad de la información no juegue un rol importante en la política militar.

3.2 Política de Seguridad Comercial

En el entorno comercial es importante prevenir la divulgación no autorizada de información, pero no lo es tanto como el prevenir la modificación no autorizada de la misma [6]. De hecho, el objetivo principal de esta política, más que la privacidad, es la integridad de la información.
La seguridad de un sistema posee la propiedad de integridad sí, los datos manipulados por éste no son alterados o destruidos por usuarios, entidades o procesos no autorizados.

Existen dos mecanismos esenciales para el control de integridad de la información:

- **La transacción bien formada.**
  Consiste en que los usuarios no deben manipular los datos de manera arbitraria, sino que únicamente en las formas que preserven o aseguren la integridad.

- **La separación de funciones.**
  Consiste en separar las funciones de los diferentes empleados para realizar de forma correcta una sola tarea, es decir, las operaciones son separadas en varias subpartes, cada una de éstas es ejecutada por una persona diferente, por lo que cada usuario tendrá derecho sólo sobre un conjunto específico de tareas y la asignación de éstas a usuarios deberá ser supervisada.

El método anterior es efectivo sólo bajo la suposición de que no exista confabulación entre los empleados.
En el ambiente de integridad comercial, el poseedor de una aplicación y los controles de procesamiento de datos implementados para la organización, son los responsables de asegurar que todos los programas sean transacciones bien formadas. Al igual que en el ambiente militar, se usa un "staff" que se responsabiliza de que los usuarios puedan ejecutar transacciones de tal forma que cumplan con el modelo de separación de funciones. El sistema asegura que el usuario no pueda evitar estos controles, por lo que se considera un control mandatorio y no un control discresionario.

3.2.1 Un modelo de integridad

Para la adecuada implementación de este modelo, descrito en [6], antes que nada se deben identificar y etiquetar aquellos conjuntos de datos sobre los que se va a aplicar el modelo. Estos conjuntos son llamados CDIs (Constrained Data Item) cuando son datos restringidos (íntegros) y UDIs (Unconstrained Data Item) cuando no lo son.

Las siguientes 9 reglas, definen un sistema que hace respetar una política
de integridad.

1) Debe probarse que todos los IVP's (Procedimientos de Verificación de Integridad) revisen correctamente la integridad de los CDI's, es decir, deben asegurar que todos los CDI's estén en un estado válido (íntegro) en el momento que está corriendo el IVP. Esta prueba puede realizarse manualmente o con las herramientas adecuadas (certificación).

2) Todos los TP's (Procedimientos de Transformación) deben ser validados, esto significa que si un TP toma como entrada un CDI íntegro debe entregar en su salida un CDI íntegro (esto garantiza que un sistema pase de un estado seguro a otro también seguro). El administrador de seguridad debe construir para cada TP la lista de CDI's para los cuales el TP ha sido validado. Todos los TP's deben certificarse para ser validados.

3) El sistema debe controlar que todo CDI manipulado por un TP esté contenido en la lista de control de acceso de dicho TP.

4) El sistema debe mantener una lista de relaciones de la forma (UserID,Tp_i(CDI_a, CDI_b, CDI_c...)), la cual describe a un usuario, una lista de TP's y los objetos de datos a los que los TP's pueden referirse en representación de dicho usuario. Es decir, el sistema debe controlar que toda ejecución de un TP sobre un CDI, solicitada por un usuario, sea autorizada.

5) El administrador de seguridad debe construir, para cada usuario, la lista de TP's que dicho usuario puede ejecutar. Cada TP en la lista es acompañado por los CDI's que puede manipular de parte del usuario. Estas listas deben garantizar la separación de funciones.

   Nota: Para mantener la integridad de los CDIs, el sistema debe asegurar que sólo un TP puede manejarlos, o al menos, que para realizar su función, un TP debe manejarlos.

6) El sistema debe identificar y autentificar la identidad de cada usuario que intenta ejecutar un TP.

   Esta regla es de gran importancia para los sistemas tanto comerciales como militares, ya que de cualquier forma estas dos clases de sistemas usan la identidad del usuario para asegurar muchas políticas diferentes. La política más relevante en el contexto militar, como se describe en el "Orange Book", se basa en niveles y categorías, mientras que la política comercial esta basada en la separación de responsabilidades entre dos o más usuarios.

7) Debe certificarse que todos los TP's escriben en modo agregar, sobre un CDI particular, toda la información necesaria a una auditoría posterior.

8) Cualquier TP que toma un UDI (datos sin restricciones, es decir que no están cubiertos por la política de seguridad) como un valor de entrada, debe ser certificado para ejecutar sólo transformaciones válidas, es decir, deberá transformar el UDI en un CDI. Un UDI puede ser, por ejemplo, algún dato ingresado por teclado, del cual no se conoce su integridad.

9) Sólo el usuario permitido para certificar entidades puede modificar la lista de tales entidades asociadas a los TP's. Todo usuario certificador no tiene autorización de ejecutar los TP's que el ha certificado.

Sería muy bueno si los mismos mecanismos pudieran tener los mismos objetivos para los sistemas comercial y militar, es decir, que se tratara con igual importancia a la confidencialidad como a la integridad de la información.
Sin embargo, hay que destacar que los mecanismos que permiten el control de la divulgación son muy diferentes a otros que permiten el control de la integridad.
Algunas de las diferencias entre políticas orientadas a confidencialidad y políticas orientadas a integridad son:

1) El mecanismo de integridad difiere en gran parte del mecanismo de mando empleado en el sistema militar descrito en el Orange Book.

- Primero: porque con estos controles de integridad, los datos no necesariamente deben estar asociados con un nivel particular de seguridad.

- Segundo: Un usuario no está autorizado para leer o escribir ciertos datos, pero si para ejecutar ciertos programas sobre ciertos datos.

2) Con los controles multinivel, un usuario es restringido a los datos que puede leer o escribir, mientras que, con los controles de integridad comercial, el usuario es restringido a los programas que puede ejecutar y la manera en la cual puede leer o escribir está implícita en las acciones de estos programas.

En la introducción del Orange Book se proponen los siguientes criterios, con los que se comparan los requerimientos fundamentales para la seguridad de computadoras.

1) El sistema debe autentificar e identificar por separado a cada usuario, para que las acciones puedan controlarse y auditarse.

2) El sistema debe asegurar qué conjuntos de datos específicos puedan manipularse sólo por un conjunto restringido de programas. Los controles centrales de datos deben asegurar que estos programas respeten la regla de "transacción bien formada".

3) El sistema debe asociar a cada usuario un conjunto válido de programas a ser ejecutados y los controles centrales de datos deben asegurar que estos conjuntos respeten la regla de "separación de derechos".

4) El sistema debe mantener y auditar los registros que cada programa ejecuta y el nombre del usuario autorizado.

A pesar de las diferencias, existen dos requerimientos en común en los ambientes militar y comercial:

1) El sistema debe contener mecanismos para asegurar que éste cumpla con su función.
2) Los mecanismos en el sistema deben ser protegidos contra atentados o cambios no autorizados.

Estos dos requerimientos, los cuales aseguran que el sistema realiza lo que se le impone que haga, son una parte integral de cualquier política de seguridad.

3.3 Política de Seguridad Financiera

Brewer y Nash [7] proponen la política Financiera (también conocida como la Muralla China), la cual combina la discreción con la fuerza de los controles de la seguridad mandatoria. Esta política es la más significativa y requerida en las operaciones de organizaciones del mundo financiero.
La base de la política de la Muralla China es permitir a las personas el acceso a la información que no esté en conflicto de intereses con alguna otra información que ellos ya posean. También se puede tener acceso a la información que el usuario haya accesado previamente. Esto significa que la información está en el Conjunto de Datos de la Compañía.
Así, el acceso a la información es concedido sólo si el objeto solicitado por un sujeto:

- Está en el mismo Conjunto de Datos de la Compañía de un objeto que ya ha sido accesado por el sujeto, es decir, que está dentro de la muralla (seguridad simple) o

- Pertenece a una Clase de Conflicto de Interés totalmente diferente

3.3.1 Organización de los datos

Toda la información corporativa es almacenada en un sistema jerárquico. Existen tres niveles :

1) Nivel más bajo : aquí se consideran partidas individuales de información (objeto) cada una concerniente a una sola corporación.

2) Nivel intermedio : se agrupan todas las partidas pertenecientes a la misma corporación en algo que se le llama Conjunto de Datos de la Compañía.

3) Nivel más alto : aquí se agrupan todos los Conjuntos de Datos de las Compañías cuyas corporaciones están en competencia. A dichas agrupaciones se les llaman "Clases de Conflicto de Interés". Un ejemplo de los nombres de las clases de conflicto de interés se puede encontrar en los sectores: financiero, petroquímica, salud, educación, etc.

Se asocia cada objeto con el nombre del Conjunto de Datos de la Compañía y al nombre de la Clase de Conflicto de Interés a los cuales pertenece.

De ahora en adelante, se hará referencia al Conjunto de Datos de Compañía como CDC y a la Clase de Conflicto de interés como CCI.

Con el fin de ejemplificar consideremos los conjuntos de datos del Banco A, de la Compañía de Petróleo A y la Compañía de Petróleo B; un usuario nuevo puede accesar a cualquier conjunto de datos que desee ya que este usuario no posee ninguna información y por lo tanto tampoco existe conflicto de interés, aunque más tarde estos pueden surgir.
Suponiendo que el usuario accesa en primer lugar el conjunto de datos de la Compañía de Petróleo A; entonces se dice que ese usuario posee información que concierne a esta Compañía. Más tarde él pide accesar al conjunto de datos del Banco A. Esto se permite por que el conjunto de datos del Banco A y de la Compañía de Petróleo A pertenecen a diferentes clases de conflicto de intereses, y entonces no existe conflicto. Ahora, sí se realiza una nueva petición de acceso sobre el conjunto de datos de la Compañía de Petróleo B, esta petición debe ser negada porque existe un conflicto entre el conjunto de datos que se pide (la Compañía de Petróleo B) y aquel conjunto de datos que ya se posee (Compañía de Petróleo A).

Se puede observar que, inicialmente, el usuario tuvo completa libertad para accesar el objeto de su preferencia, y con la selección inicial realizada se creó una *Muralla China* para el usuario alrededor del conjunto de datos. En realidad no importa si el conjunto de datos de la Cía. de Petróleo A fue accesado antes o después que el conjunto de datos del Banco A; pero si la Compañía de Petróleo B hubiese sido accesada antes de la petición de accesar al conjunto de datos de la Compañía de Petróleo A, serían muy diferentes las restricciones y la muralla que se formaría. En este caso sería negado el acceso al conjunto de datos de la Cía. de Petróleo A y lo que el usuario podría poseer sería la Cía de petróleo B y el Banco A . La política de la muralla China es una combinación de elección libre y control mandatorio [7].

Los teoremas de esta política son:

T1. Una vez que un sujeto ha accesado un objeto los únicos otros objetos accesibles por ese sujeto descansan dentro del mismo conjunto de datos de la Compañía o dentro de una diferente Clase de Conflicto de Interés.

T2. Un sujeto puede tener acceso, a lo más a un Conjunto de Datos de Compañía en cada Clase de Conflicto de Interés.

T3. Si para una Clase de Conflicto de Interés X hay X_y Conjuntos de Datos de Compañía, entonces, el número mínimo de sujetos que permitirán que cada Conjunto de Datos de Compañía sea accesado por al menos un sujeto es X_y.

T4. El flujo de información "no saneada" está confinado a su propio Conjunto de Datos de Compañía; la información "saneada" puede fluir libremente a través del sistema.

La información saneada es aquella que ha pasado una censura, y no compromete a una empresa. La información no saneada es aquella que es confidencial.

3.3.2 Violaciones indirectas

Dentro de esta política se pueden suponer violaciones indirectas. Supongamos que se tienen dos sujetos, el usuario A y el usuario B, entre los dos tienen acceso a los tres conjuntos de datos, a la Cía. de Petróleos A, a la Cía. de Petróleos B y al Banco A; el usuario A tiene acceso a la Cía. de Petróleos A y al Banco A, mientras que el usuario B tiene acceso a la Cía. de Petróleos B y al Banco A. Si el usuario A lee información de la Cía. de Petróleos A y la escribe al Banco A entonces el usuario B puede leer la información de la Cía. de Petróleos A. Esto no debe estar permitido debido al conflicto de interés entre la Cía. de petróleos A y la Cía. de petróleos B. Por lo tanto para prevenir tales violaciones se debe cumplir lo siguiente:

El acceso en escritura será permitido sólo si:

- El acceso es permitido por la seguridad simple.

- Ningún objeto puede ser leído si está en un Conjunto de Datos de Cía. diferente a aquel de donde el acceso en escritura es solicitado y dicho objeto contiene información no saneada.

En general, se ha demostrado que la representación formal de esta política permite que el acceso a los bancos de datos de una Cía. sea posible sólo si no representa competencia, es decir que no genere conflicto de interés para la Cía. que realiza el acceso. Todas las Compañías que no compiten, están dentro de la muralla y pueden compartir su información.

4 ENFOQUES DE SEGURIDAD MULTIDOMINIO

En este capítulo se presentan los enfoques más importantes sobre el tema de seguridad multidominio. Se abordan aspectos importantes que sirven como punto de comparación con el enfoque que se utiliza en este trabajo y que se utilizó para el desarrollo del sistema.

4.1 EL ENFOQUE DE ECMA

En este capítulo se presenta una descripción de los enfoques más importantes sobre el tema de Seguridad Multidominio. Esta descripción nos permite conocer cómo se encuentra la teoría al respecto y poder tener elementos para valorar la posibilidad de implementación de un sistema controlador de políticas de seguridad con un enfoque mandatorio.
Actualmente, se considera que una política de seguridad debe ser dividida en varias subpolíticas de acuerdo a la función de seguridad que deben cumplir, por ejemplo: confidencialidad, integridad y negación de servicio [4][11].

ECMA [8] considera dos conceptos fundamentales en su enfoque: dominio de seguridad y facilidad de seguridad. Este último concepto se refiere a técnicas de modelado para desarrollar y describir la seguridad en sistemas abiertos, en otras palabras, podemos decir que es un conjunto lógicamente relacionado de funciones de seguridad que permiten el modelado de un sistema seguro.

Para que una interacción entre dominios sea segura, se necesitan controles predefinidos de seguridad [9]. Esta situación se dificulta porque en la realidad existe una diversidad de políticas. La razón de esto es porque cada dominio hace cumplir una política diferente además de que tienen diferentes autoridades de seguridad.
La razón de que exista una diversidad de políticas se debe a que cada organización tiene su propio entorno. En dicho entorno se considera lo siguiente :

- Cada nación tiene sus propias normas y estándares de seguridad
- Existen diferentes sectores socioeconómicos, por ejemplo, sector comercial, de salud, financiero, etc.
- Gran variedad en productos de software y hardware.
- Diferencias en la interpretación de las políticas de seguridad.
- Uso de diferentes mecanismos de seguridad.

La integridad y la confidencialidad se pueden entender bien en el contexto de multidominio, mientras que el control de acceso y los servicios de autentificación son mucho más complejos.
Por otra parte, ECMA define un marco para seguridad en sistemas distribuidos. El TR/46 [8] presenta las bases para que exista una cooperación entre diferentes dominios de seguridad.

En un dominio de seguridad pueden existir subdominios. En estos subdominios es posible que se aplique total o parcialmente la política del dominio en el cual están contenidos. Las relaciones entre dominios y subdominios son las siguientes:

- Relaciones de sub-dominio. Un dominio de seguridad es un sub-dominio de otro que es más general.

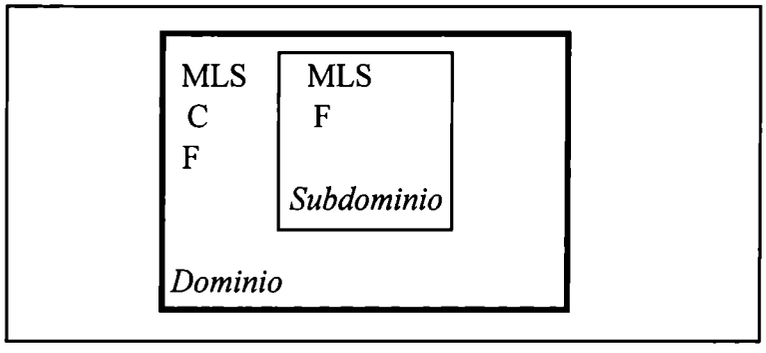

Fig. 4.1.1 Relación de sub-dominio

- Relaciones par. Dos dominios son pares si no son sub-dominios de otro.

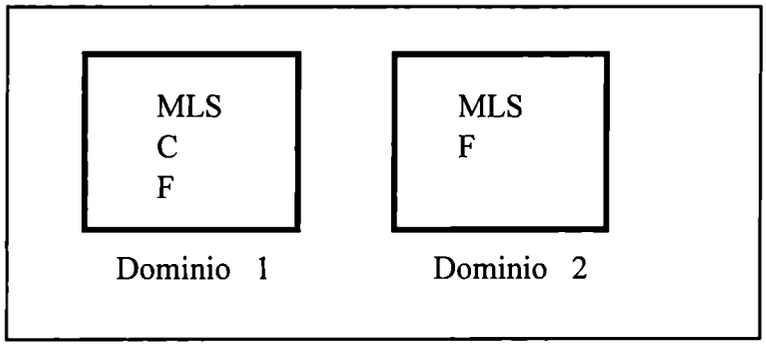

Fig. 4.1.2 Relación par

Dentro de estas relaciones puede suceder que un subdominio forme parte de la política global, o bien, que no forme parte de ella.
A esto se le conoce como "delegación exclusiva" y "delegación no exclusiva" respectivamente.

Hay otro tipo de relaciones que ECMA define como "relaciones par". Dentro de estas relaciones tenemos los siguientes tipos:

- Dominios pares autónomos. Cada dominio tiene su propia política de seguridad y no hay un control común sobre ellas.

- Dominios pares no autónomos. Cada dominio tiene su propia política de seguridad y hay un control común sobre ellas.

El TR/46 también incluye la *facilidad interdominio* para modelar las interacciones de los dominios de seguridad. La interpretación de los atributos de seguridad y el control sobre el intercambio de datos seguros, es la principal tarea de esta facilidad. Para ello se necesita un traductor intermedio, que pueda interpretar la sintaxis.

Las funciones de una política de seguridad pueden identificarse en lo siguiente:

- Dominio de seguridad en sistemas distribuidos. En este caso la política de seguridad es diseñada para un ambiente distribuido.

- Domino de seguridad local. En este caso la política de seguridad es diseñada para un entorno local.

- Dominio de seguridad en aplicaciones. Aquí, el alcance de la política es una aplicación en particular.

Es importante hacer notar que el alcance de este enfoque contempla a los sistemas distribuidos. Se puede hacer una combinación de los puntos anteriores, por ejemplo, una aplicación con su propia política de seguridad puede estar contenida en un dominio local, el cual a su vez puede estar contenido en un dominio para sistemas distribuidos.
ECMA propone un servicio interdominio constituido por una combinación de facilidades. Los dominios usan estos servicios para establecer asociaciones seguras. Este servicio "mapea" los atributos de seguridad y los identificadores de autoridad. Cualquier objeto que vaya de un dominio a otro debe especificar sus atributos de seguridad con una sintaxis predeterminada. El "mapeo" de estos atributos hace que el rendimiento no sea el óptimo y como consecuencia de ello, la comunicación en tiempo real no es posible.

Otro caso que contempla este enfoque es en el que interactúan dominios mutuamente confiables. En estas circunstancias cada autoridad de dominio posee la misma forma de traducción de atributos.

4.2 MULTIPOLITICAS

Hosmer ha introducido algunos conceptos nuevos de seguridad Multidominio [14]. En general los componentes del modelo de Hosmer son: objetos, sujetos, operaciones permitidas, "lattice" de información de seguridad y monitor de referencia.
El problema de la comunicación segura entre dominios diferentes es un tema que requiere investigación futura. Los puntos inmediatos que se tienen que abordar, según Hosmer, son los siguientes:

- Las políticas deben ser comparables, para ello se requiere un lenguaje común para describirlas.
- No existe una guía de integración.
- No hay herramientas para comparar políticas de seguridad.

Las fallas que Hosmer ha identificado en los conceptos de seguridad actuales son:

- No son flexibles. Esto se refiere a que si una política de seguridad es modificada, se tiene que evaluar nuevamente el sistema completo porque existe la posibilidad de crear un hueco de seguridad.
- El intercambio de información en tiempo real entre políticas diferentes es difícil o imposible. Este punto está en concordancia con el enfoque de ECMA presentado anteriormente.
- Las políticas enfocadas a hacer seguro un solo dominio no son realistas. En la realidad existen múltiples políticas de seguridad.

Hosmer propone nuevos conceptos en seguridad para ser desarrollados:

- Construcción bottom-up. Este tipo de construcción para políticas de seguridad evitaría tener que reevaluar el sistema en términos de seguridad cuando una política sea modificada.
- Separar las políticas de seguridad de los mecanismos que las implementan.
- Facilitar la integración de componentes de seguridad. Los nuevos conceptos deben poseer estándares y modelos para adaptar variaciones en las políticas.
- Permitir la operación paralela de políticas contradictorias.

Hosmer intenta cubrir algunas de estas necesidades con su modelo conceptual. En este modelo se permite a los usuarios incluir sus propias políticas sin reevaluar el sistema completamente. La integración preserva la clasificación original, es decir, no hay traducción de atributos, y además es posible la implementación paralela de diversas políticas (ver figura 4.2.1).

Los elementos de este modelo son:

- Dominio de seguridad
- Código del dominio de seguridad
- Múltiples políticas de seguridad
- Segmentos de etiqueta
- Metapolíticas
- Mecanismo que hace respetar la política

En este modelo se permiten diferentes autoridades de dominio autónomas, además los administradores de los dominios no pueden modificar su propia política.

Todos los objetos tienen un identificador de dominio. De esta manera es posible conocer cual es la política asociada a ese objeto en particular.

Los segmentos de etiquetas, describen todos los atributos de seguridad del objeto. Existe un segmento para cada política de seguridad (segmento de integridad, segmento de confidencialidad, etc.).

Las metapolíticas especifican un conjunto de reglas para las relaciones e interacciones entre políticas.

Hosmer y Abrams proponen la *máquina multipolítica* [18][19], la cual intenta resolver los problemas asociados con la interacción de políticas. Esta máquina está compuesta por :

- Código de la política del dominio. Este código determina la política que se aplicará a los objetos y sujetos.

- Intérprete de código de dominio. Maneja las etiquetas de seguridad de forma adecuada para que se cumplan las políticas.

- Mecanismo que hace respetar la política (Enforcer). Es el que se encarga de aplicar las reglas de una política particular a los objetos y sujetos del sistema.

- Metapolítica. Se encarga de coordinar las políticas de seguridad.

Para implementar la máquina multipolítica, aparecen algunas de las opciones de implementación propuestas en [22]. Estas propuestas son:

- Reglas basadas en el enfoque de Hosmer. Lo importante en este punto es que las autoridades de los dominios pueden modificar sus propias políticas sin afectar la seguridad global del sistema.

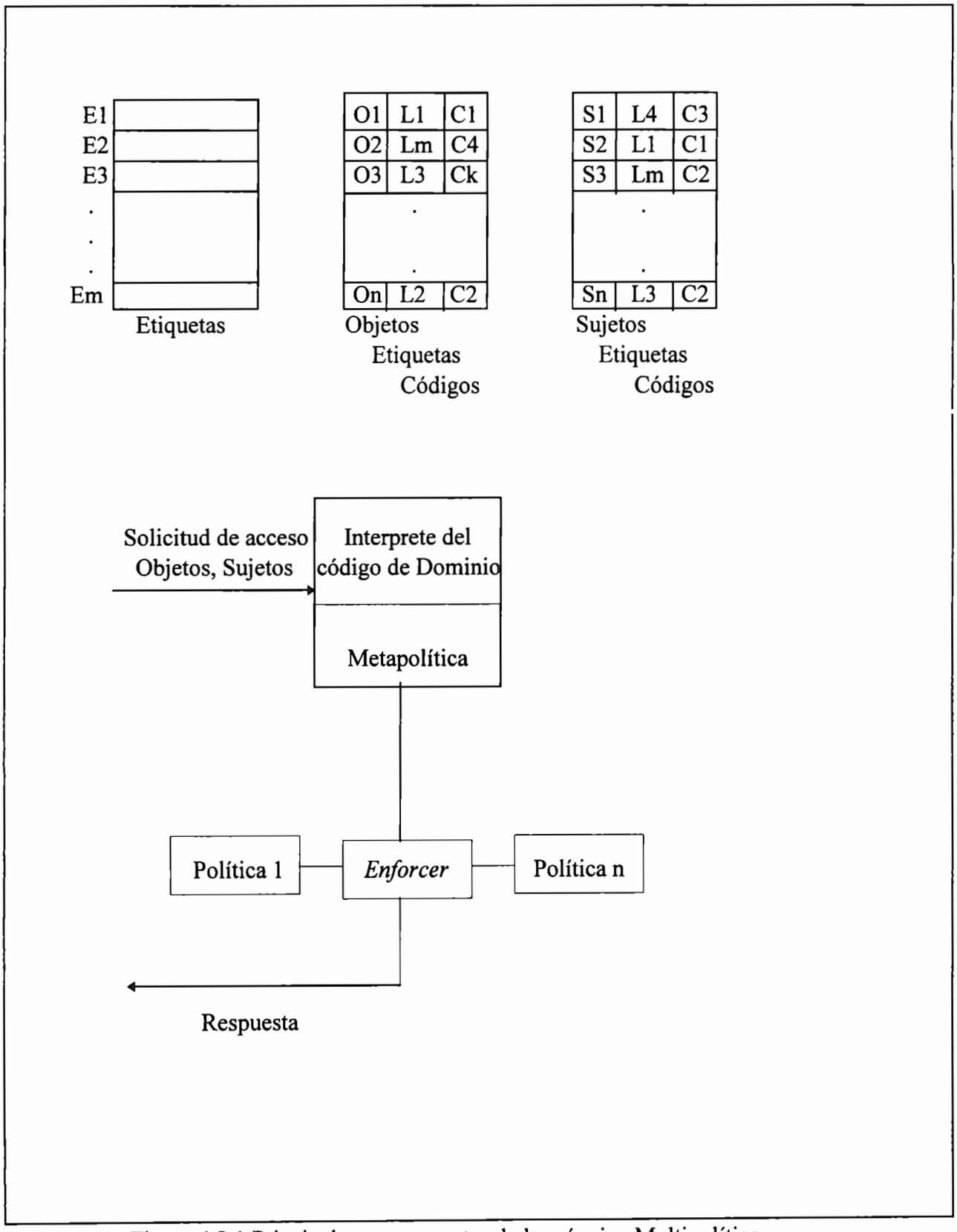

Figura 4.2.1 Principales componentes de la máquina Multipolítica

- Procesadores paralelos. En este enfoque una política de seguridad y el mecanismo que fuerza a que sea cumplida, se encuentran en hardware (en un VLSI).

Algunas de las funciones de las metapolíticas de Hosmer son:

- Proveen flexibilidad para la adición, modificación y supresión entre políticas y subpolíticas.
- Coordinan las interacciones y relaciones entre políticas y subpolíticas.
- Definen las operaciones permitidas en la frontera entre dominios.
- Resuelven las incompatibilidades y contradicciones.

4.3 ENFOQUE DE L.J. LA PADULA

L.J. La Padula [17] presenta un enfoque para modelar formalmente el control de acceso . Uno de los objetivos de este enfoque es el de permitir el desarrollo de las políticas de seguridad de acuerdo al las necesidades del usuario, es decir, se pueden ir configurando las políticas conforme el usuario lo requiera. Una política que es construida de esta manera no tiene que ser evaluada otra vez cuando exista una modificación o la creación de una nueva subpolítica.

Otro objetivo consiste en proporcionar procesos confiables. Para que estos procesos confiables existan en la política de seguridad, este enfoque propone la separación de los controles de seguridad de los mecanismos que hacen cumplir la política.

La Padula separa las funciones del sistema de las reglas de acceso [17]. Las solicitudes de acceso son recibidas directamente por las reglas de acceso y no por las funciones del sistema. Todas las reglas son verificadas para cualquier solicitud de acceso. Si tan sólo una regla retorna un NO, entonces el acceso es negado. Cuando todas las reglas retornan un SI, entonces el acceso es permitido. Cuando no se especifique un ELSE en el código de las reglas de control de acceso, entonces el valor retornado será un NO si la condición no se cumple. Las reglas en este enfoque toman la siguiente forma (las reglas suponen que un *sujeto* realiza una solicitud de acceso a un *objeto*) :

- Cuando una regla será aplicada a una política particular:

IF <la solicitud no concierne a esta política> or
  <la solicitud concierne y satisface el criterio
  de esta política >

THEN <return code=YES; eventualmente:
actualización de la información de seguridad>

- Cuando la regla será aplicada a todas las políticas:

IF <la solicitud satisface el criterio de la política global>
THEN <return code=YES; eventualmente:
actualización de la información de seguridad >

- Cuando una solicitud particular es necesitada para actualizar la información de seguridad:

IF <la solicitud es específica>
THEN <return code=YES;
actualización de la información de seguridad>
ELSE <return code=YES>

Como ejemplo consideremos cuatro usuarios que tienen una política de seguridad diferente entre sí. El usuario 1 tiene una política de control de acceso discresionario (DAC), el segundo una de control de acceso mandatorio (MAC), el tercero una de control funcional (FC) y el cuarto tiene una (DAC o MAC) y FC.

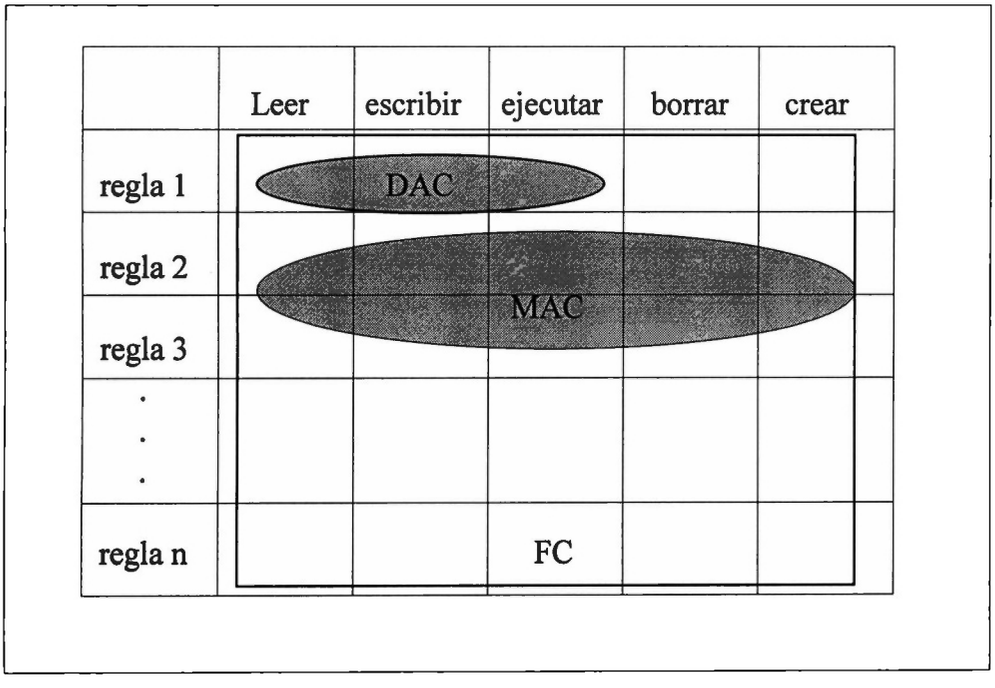

Fig. 4.3.1 Política de control de acceso

Continuando con este ejemplo, consideremos que el criterio DAC es utilizado por la regla 1 para solicitudes de lectura, escritura y ejecución. El criterio de MAC es utilizado por las reglas 2 y 3 para solicitudes de lectura, escritura, ejecución, borrar y crear. Por último el criterio FC es utilizado por todas las reglas. En la figura 4.3.1 aparece este ejemplo de manera gráfica. El ejemplo en pseudocódigo es como sigue:

- Regla 1 (DAC)

IF <not DAC> or
  <<solicitud in {leer, escribir, ejecutar}> and
   <(sujeto,solicitud) in lista-acceso(objeto)>>
THEN <return=YES>

- Regla 2 (MAC)

IF <not MAC> or
  <<solicitud in {leer,ejecutar,borrar}> and
   <clase(sujeto) domina a clase(objeto)>> or

  <<solicitud in {escribir}> and
   <clase(objeto) domina a clase(sujeto)>> or
   <solicitud in {crear}>
THEN return=YES

- Regla 3 (MAC)

IF <solicitud in {crear}>
THEN <return=YES>;
    <clase(objeto) ← clase(sujeto)>
ELSE <return=YES>

- Regla 1..n (FC)

IF <rol(sujeto) concuerda con función(objeto)>
THEN <return=YES>

Este enfoque se compone de dos partes; una de ellas son las funciones del sistema (interfase) y la otra son las reglas de seguridad. Para ilustrar este enfoque La Padula usa la interface del "modelo universal" de J. Millen [17]. Este es un modelo de máquina de estados para control de flujo de información.

4.4 LÓGICA DE SEGURIDAD

Un enfoque formal para el razonamiento de seguridad de sistemas es el de Lógica de Seguridad (SL), el cual fue propuesto por Glassgow, McEwen y Panangaden [20]. Los elementos fundamentales son: conocimiento, permiso y obligación. Con estos elementos es posible modelar la seguridad en sus aspectos de integridad y confidencialidad. Otro punto importante de este enfoque es que permite combinar políticas de seguridad. Utiliza la noción de *mundos posibles* para definir la lógica de conocimiento [21].
En SL, dos mundos o estados son indistinguibles si ellos pertenecen a la misma relación de equivalencia. Un mundo modela una fórmula si ésta es verdadera en este el mundo. Esto se representa de la siguiente forma:

$s \models f$

Un mundo s modela que un sujeto i conoce una fórmula f si, y sólo si, la fórmula f es modelada por otro mundo s' que es equivalente a s.

$s \models K_i f$

La notación $K_i f$ significa que un sujeto i conoce que la fórmula f es verdadera.

Se consideran tres operadores lógicos temporales:

$\forall\Box$ (siempre)

$\forall\Diamond$ (eventualmente)

$\exists\Diamond$ (algunas veces)

$s \models \forall\Box f$ si y sólo si f es verdadera para todos los mundos y para todas las rutas empezando en s.

$s \models \forall\Diamond f$ si y sólo si f es verdadera para un particular mundo en todas las rutas empezando en s.

$s \models \exists\Diamond f$ si y sólo si f es verdadera para un mundo particular en una ruta particular empezando en s.

4.4.1 Concepto de fórmula bien formada

Una fórmula bien formada incluye:

If f y $g \in W$ entonces también son $\neg f$, $f \wedge g$, $f \vee g$, y $f \Rightarrow g$.
donde W es el conjunto de fórmulas bien formadas.

- If $f \in W$ entonces se cumple que $K_i f$ y los operadores temporales se aplican a f: $\forall\Box$ f, $\forall\Diamond$ f, $\exists\Diamond$ f.

4.4.2 Permiso y obligación

El concepto de permiso permite llevar a cabo el control sobre la confidencialidad, mientras que el de obligación lo hace para la integridad. Una aserción negativa que previene que ciertos flujos ocurran (permiso) es conocida como una propiedad de "seguridad". Por otra parte una aserción positiva que permite que ciertos flujos ocurran (obligación) es conocida como una propiedad de vivacidad.
Estos conceptos se asemejan al de Clark y Wilson [2] en sus "transacciones bien formadas". Según estos autores, sí se puede garantizar que ciertos flujos específicos ocurran no es necesario prohibir explícitamente los demás flujos que puedan presentarse.
Continuando con la notación en SL tenemos que:

Los operadores de permiso y obligación son incluidos en la Lógica de Seguridad.

1. $OK_i f$ significa que *el sujeto i está obligado a conocer f*

2. $PK_i f$ significa que *el sujeto i tiene permiso de conocer f*

3. OBL_i es el conjunto de fórmulas que el sujeto i está obligado a conocer.

4. PER_i es el conjunto de fórmulas que al sujeto i se le permite conocer.

   De lo anterior se desprende que :
   If $f \in OBL_i$ entonces $f \in PER_i$
   Si el sujeto *i* está obligado a conocer *f*, entonces debe tener permiso de conocerla.

5. $\forall f \in W, s \models PK_i f$ si y solo si $s \models f$ and $f \in PER_i$

Otros valores posibles son:

$\forall f \in W, s \models PK_i f$ si y sólo si $s \models f$ and

$((f=g \wedge h$ and $(s \models PK_i g$ and $s \models PK_i h))$ or

$(f=g \vee h$ and $(s \models PK_i g$ or $s \models PK_i h))$

Por otra parte, un sujeto está obligado a conocer una fórmula sólo si ésta es verdadera y está contenida en su conjunto de obligación OBL.

$\forall f \in W, s \models OK_i f$ si y solo si $s \models f$ and

$f \in OBL_i$

No hay otros valores verdaderos para la obligación. Analizando el caso anterior para *permiso*, concluimos que esto no es posible para la *obligación* ya que al tener permiso de conocer una fórmula no implica tener la obligación de conocerla.
La integridad y confidencialidad pueden ser definidas utilizando los conceptos de permiso y obligación conjuntamente con el de conocimiento:

Confidencialidad:

If $s \models K_i f$ then $s \models PK_i f$ (Conocimiento y permiso)

Integridad:

If $s \models OK_i f$ then $s \models \forall\Diamond K_i f$ (Conocimiento y obligación)

En otras palabras, la propiedad de confidencialidad dice que un sujeto ***i*** puede conocer un secreto ***f***, sólo si tiene permiso de conocer el secreto ***f***.

La propiedad de integridad indica que un usuario ***i*** que está obligado a conocer una información ***f*** en un estado ***s*** debe, eventualmente (en algún momento), conocer esta información.

Una política de seguridad puede ser definida en términos de *permiso y obligación*. Por ejemplo, la política multinivel puede ser definida de la siguiente manera [20]:
Para todas las fórmulas bien formadas f, un sujeto i, $f \in PER_i$, existe confidencialidad multinivel si, y sólo si el nivel de confidencialidad de f es dominado por el nivel de confidencialidad de i. La fórmula f puede ser interpretada como información sensible.
Como podemos observar en el párrafo anterior, para determinar la confidencialidad no es necesario un conjunto de obligaciones para i (OBL_i).
Haciendo referencia a la política comercial de Clark y Wilson, podemos decir que aquí cada usuario o sujeto i tiene asociado un conjunto de obligaciones OBL_i. Este conjunto contiene los programas que el sujeto i tiene derecho a ejecutar. En términos de la política comercial esto se llama *transacciones bien formadas* y son las que permiten un flujo de información de una unidad íntegra a otra [20].

Como una reflexión sobre lo anterior se menciona la siguiente situación:

Un usuario i está obligado a conocer una fórmula f :

- si esta fórmula es necesaria cuando ***i*** invoca a una *transacción bien formada **j***.

- si el usuario ***i*** está obligado a conocer otra fórmula ***f'*** que es necesaria cuando ***i*** invoca a una *transacción bien formada* ***j'***, la cual contiene información que puede ser accesada por la *transacción bien formada* ***j***.

Expresando lo anterior de otra manera tenemos que para que un usuario i accese cierta información tiene que accesar primero la *transacción bien formada* asociada a dicha información. También tiene que conocer la fórmula f que permite invocar la transacción.
Si una invocación del usuario i debe producir un flujo de información desde j hasta j' (transacciones bien formadas) y este usuario es capaz de invocar a j' (es decir, este usuario está obligado a conocer la fórmula f' que lo habilita a invocar a j') entonces también está obligado a conocer a la fórmula f que invoca j.

En este enfoque es posible la combinación de políticas de seguridad. Si el resultado de esta combinación satisface las propiedades de las políticas que la componen, se dice que existe compatibilidad.
Para la combinación de políticas de seguridad en este enfoque tenemos lo siguiente:

- A un sujeto se le permite conocer una fórmula de una política combinada si y sólo si se le permite conocer la fórmula en todas las políticas originales.

- Un sujeto está obligado a conocer una fórmula en una política combinada si y sólo si el sujeto está obligado a conocer la fórmula en una o más políticas originales

Para la implementación del enfoque descrito en [1], y que es tratado más adelante, se utiliza una serie de combinaciones de tres políticas; Multinivel, Comercial y Financiera. La combinación de dos o más de ellas conserva las propiedades de las políticas originales. Debido a lo anterior, si se intenta accesar información de una política combinada compuesta por ejemplo, por Multinivel y Comercial, desde una política multinivel, la comunicación es posible desde el punto de vista de consistencia de políticas. Para este ejemplo, es posible que la comunicación no sea permitida si los niveles de autorización del origen y el destino no son los adecuados, pero esto corresponde exclusivamente a la aplicación de la política MLS.

En la combinación de políticas se deben respetar dos propiedades fundamentales:

- Consistencia. Esta propiedad consiste en que no existan contradicciones entre una política X y una política Y, donde X esté contenida en Y, es decir, donde Y es una combinación que incluye a X como un componente o política original.

- Compatibilidad. Dos políticas son compatibles si todas las propiedades de una de ellas implican a todas las propiedades de la otra.

4.5 ÁLGEBRA DE SEGURIDAD

McLean [23] propone un enfoque donde se permite la creación y comparación de modelos de seguridad. Aquí se permite la comparación y creación de modelos de seguridad a partir de otros ya existentes. Este enfoque incluye un control de acceso mandatorio.
La característica principal de este enfoque, es la habilidad que tienen algunos sujetos para modificar los niveles de seguridad.

El modelo inicial es ampliado para incluir reglas de n-personas. El término n-personas, se refiere a que son necesarios varios individuos para realizar alguna tarea. El primer caso o modelo inicial, es un álgebra booleana de modelos, mientras que la forma ampliada toma una forma de "lattice" distributivo. Con las reglas de n-personas es necesario que los controles de acceso sean discresionarios. En los dos casos mencionados se utiliza el modelo MLS. Las políticas de seguridad dentro de este enfoque, son llamadas modelos, que a su vez son máquinas de estado que permiten la transición de estados seguros.
Cuando en una transición de un estado s a otro s' el nivel de seguridad cambia, entonces el sujeto que solicitó la transición ha sido habilitado para realizar la modificación.

Los elementos para estos modelos son los siguientes:

S = conjunto de sujetos

O = conjunto de objetos

L = lattice de niveles de seguridad

A = conjunto de modos de acceso

C_s es una función que regresa el conjunto de sujetos que pueden modificar el nivel de seguridad de un sujeto en particular.

C_o es una función que regresa el conjunto de sujetos que pueden modificar el nivel de seguridad de un objeto en particular.

A cada par en C (C_s,C_o) le corresponde un modelo. El modelo de seguridad simple de Bell y La Padula BLP, es el que tiene el nivel de restricción más bajo, es decir, donde todos los sujetos pueden modificar los niveles de seguridad.

$\forall x \in S$ and $y \in O$, $C_s(x)=C_o(y)=S$

Existe un modelo en BLP más restrictivo :

$\forall x \in S$ and $y \in O$, $C_s(x)=C_o(y)=SSO$

En este modelo no se permite que ninguna modificación de los niveles de seguridad por usuarios comunes. El único que puede hacer modificaciones es SSO quien es el oficial de seguridad del sistema.

Algunos aspectos que se desprenden de los conceptos anteriores:

Un modelo M1 es "dominado" por un modelo M2 ($M1 \leq M2$) si el conjunto de sujetos que pueden modificar los niveles de seguridad en M1, es un subconjunto de los de M2.

Un modelo M1 es equivalente a M2 si y sólo si $M1 \leq M2$ y $M2 \leq M1$ si el conjunto de sujetos que pueden modificar los niveles de seguridad (de los objetos y sujetos) en M1 es un subconjunto de los sujetos que pueden modificar los niveles de seguridad en M2 y viceversa.

Si dos modelos tienen el mismo conjunto de sujetos que pueden modificar los niveles de seguridad, entonces se dice que los modelos son equivalentes.
Se pueden obtener nuevos modelos a partir de los que ya existen, por ejemplo, si M1 permite que los niveles de seguridad de los objetos puedan ser modificados sólo por sus propietarios, mientras que M2 sólo permite esto al SSO, entonces tendremos:

$M3=M1 \cup M2$

$\forall x \in S$ and $y \in O$, $C_s3(x)=C_s1(x) \cup C_s2(x) =\{SSO\}$

and

$C_o3(y)=C_o1(y) \cup C_o2(y) = \{\text{sujetos de M1}\} \cup \{SOO\}$

También la intersección y el complemento están definidos

$\forall x \in S$ and $y \in O$, $C_s(x)=S-C_s(x)$,

$C_o(y)=S-C_o(y)$

Los elementos en C_s pertenecen al conjunto potencia de S P(S) y forman un álgebra booleana para las operaciones ($\cup$ y $\cap$). Lo mismo sucede para C_o.
La relación de orden <= del álgebra booleana se utiliza aquí para la restricción de modelos.

Para el modelo M1= BLP *simple* (el menos restrictivo)

Para S, O, L, A, C y la misma política de seguridad,
$\forall x \in S$

$C_s1(x)=S$

$M1 \cap M2 = M2$

$C_s1(x) \cap C_s2(x) = C_s2(x) = \{SSO\}$

$M1 \cup M2 = M1$

$C_s1(x) \cup C_s2(x) = C_s1(x) = S$

$M1' \cup M1 = M1$ (lub)

y

$M1' \cap M1 = M3$ (glb)

Para M2 (modelo intermedio)

$C_s(x) = \{SSO\}$

$M2 \cap M3 = M3$

$C_s2(x) \cap C_s3(x) = C_s3(x) = \{\}$

$M2 \cup M3 = M2$

$C_s2(x) \cup C_s3(x) = C_s2(x) = \{SSO\}$

$M2' \cup M2 = M1$ (lub) y

$M2' \cap M2 = M3$ (glb)

donde lub es el menos restrictivo (lowest upper bound) y
glb es el más restrictivo (gratest lower bound).

Para M3 (modelo más restrictivo)

$C_s3(x) = \{\}$

$M1 \cap M3 = M3$

$C_s1(x) \cap C_s3(x) = C_s3(x) = \{\}$

$M1 \cup M3 = M1$

$C_s1(x) \cup C_s3(x)=C_s1(x)=S$

$M3' \cup M3=M1$ (lub) y

$M3' \cap M3=M3$ (glb)

Esto también se aplica a la función C_o.

La extensión del modelo anterior a uno de n-personas debe incluir algunos cambios:

$S_{np}=P(S)- \{\}$

donde S_{np} es el conjunto de sujetos que sustituye a S. Ahora las funciones C_s y C_o regresan valores en $P(S_{np})$.
Sin embargo hay que hacer una modificación para evitar que ocurran situaciones extrañas como la siguiente:
Sea {{x},{y}} un elemento de $P(S_{np})$, por lo tanto x puede modificar el nivel de seguridad de un objeto, pero no lo puede hacer con la colaboración de y.
La modificación que hay que hacer es en el rango de C_s y C_o, el cual estará contenido en otro conjunto potencia $P'(S_{np})$.

Si S={x,y} entonces

S_{np} ={{x} ,{y},{x,y}} y

$P'(S_{np})$={{}, {{x,y}},{{x},{x,y}},{{y},{x,y}},{ {x},{y},{x,y} }}

Debido a que esta extensión no satisface la operación de complemento, no es un álgebra booleana.

$P'(S_{np})$ es un lattice ordenado por la relación de subconjunto y además es distributivo porque permite las operaciones $\cup$ y $\cap$.

4.6 RESTRICCIONES EN EL SISTEMA DE VISTAS DEL USUARIO

El trabajo de McCullough [22] sobre las restricciones en el sistema de vistas del usuario contiene los siguientes puntos interesantes:

- Principio de no-interferencia [5]. Un grupo de usuarios, utilizando un cierto grupo de comandos, no interfiere con otro grupo de usuarios si los comandos que usa el primer grupo no tienen efecto sobre lo que el segundo grupo puede ver.

- Principio de no interferencia para MLS. Si todas las entradas ocultas de nivel N son suprimidas, entonces las observaciones de cualquier usuario de nivel N permanecen sin cambio.

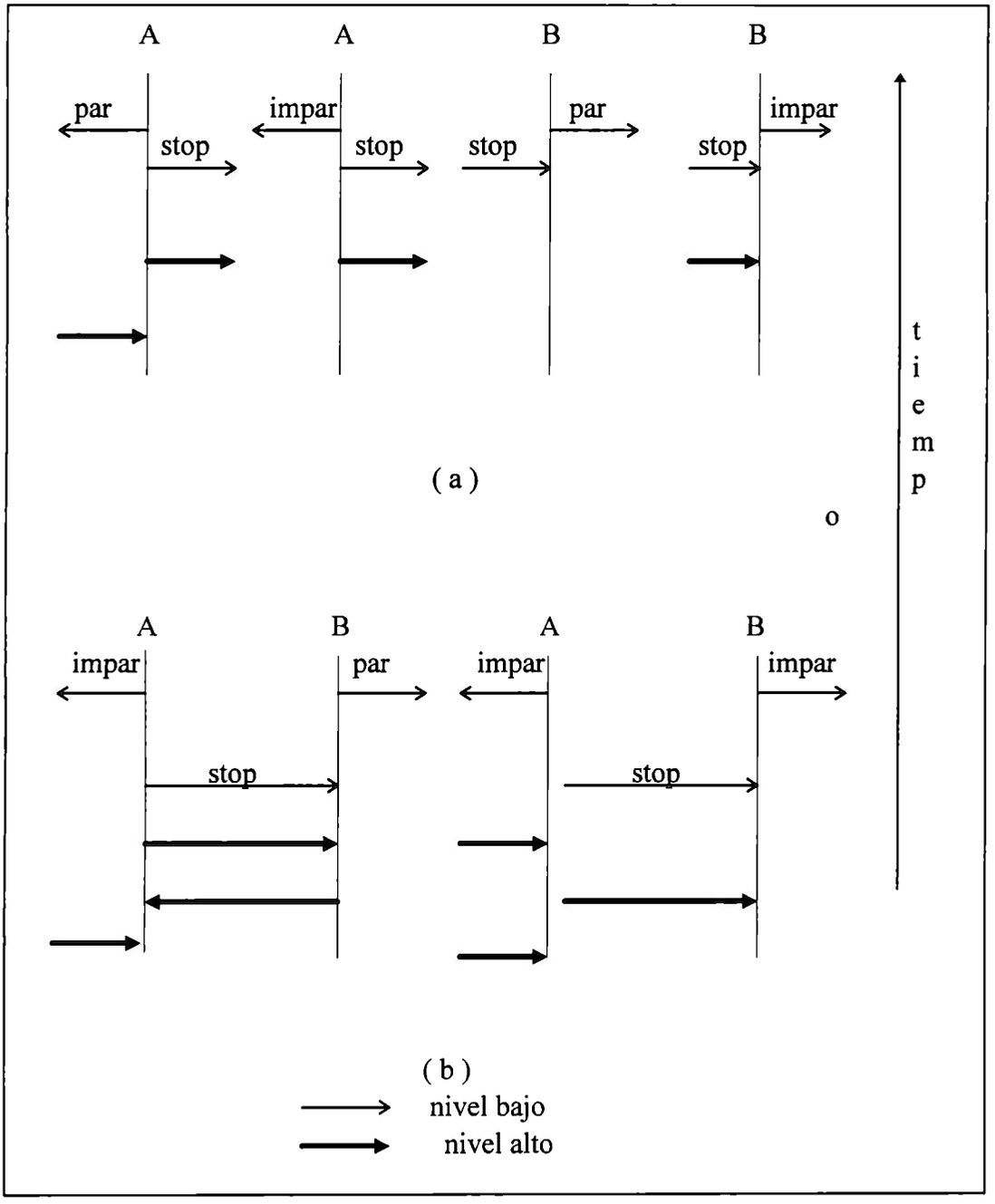

Fig. 4.6.1. Interconexión de A y B.

El principio de no interferencia es significativo para sistemas donde un conjunto particular de entradas siempre generan la misma vista para el usuario.
Deducibilidad [29]. Un sistema es deduciblemente seguro si la información contenida en los eventos en la vista para el nivel N, no revelan nada acerca de la información en las entradas ocultas desde el nivel N. Un evento puede ser una entrada o una salida.
El hecho de que no sea revelada la información de las entradas ocultas, no significa que la vista para el nivel N no sea cambiada.
Este concepto es aplicable a sistemas determinísticos y no determinísticos, por lo que es un concepto más general que el de no interferencia.

McCullough ha demostrado [22] que la deducibilidad no es preservada cuando los sistemas son interconectados.
Consideremos dos sistemas, A y B que tienen salidas de nivel bajo que pueden ser par o impar (figura 4.6.1). Estas salidas dependen del número de eventos de nivel alto (entradas o salidas) que ocurren antes de un evento denominado "stop". Este evento es una salida del sistema A y una entrada en el sistema B.
Los eventos 'par' e 'impar' sólo dan una idea a los usuarios de nivel bajo acerca del funcionamiento en el nivel alto.

Siguiendo con la figura 4.6.1, en la primera parte, podemos ver que es imposible deducir las entradas que han ocurrido en un nivel alto. En la primera traza para el sistema A se observa que hay sólo una entrada en un nivel alto y la salida en el nivel bajo es par. En la segunda traza no hay entradas en un nivel alto y la salida en un nivel bajo es impar. Para la tercera traza, ya en el sistema B, no hay entradas y la salida es par y en el último caso, la salida es impar cuando únicamente ocurrió una entrada en un nivel alto.

Los sistemas A y B analizados son deduciblemente seguros porque son sistemas independientes.

En la siguiente parte (b) de la figura 4.6.1, tenemos dos sistemas, A y B, pero ahora estos sistemas no son independientes porque están interconectados. Las salidas desde A se convierten en entradas a B y las salidas de B se convierten en entradas de A. Además A puede tener otras entradas que no provengan de B, mientras que B sólo tiene entradas que provengan de A. Se puede inferir que si las salidas en un nivel bajo son iguales entonces al menos una entrada en un nivel alto ha ocurrido en A. Las salidas en un nivel bajo en A dependen de las entradas locales en un nivel alto de A de los eventos de nivel alto en B. Expresado esto de otra manera tenemos que:
LOa = salida de nivel bajo en A.

LOb = salida de nivel bajo en B.

stop = puede ser visto como una función que regresa la paridad de eventos de nivel alto.

HIa = número de entradas locales de nivel alto en A.

HOa = número de salidas locales de nivel alto en A.

HOb = número de salidas locales de nivel alto en B.

HIba = número de entradas locales de A hacia B.

HIab = número de entradas locales de B hacia A.

La visión del usuario de nivel bajo es la siguiente :

LOa = stop(HIa+HIba+HOa)

LOb = stop(HIab+HOb)

Desde las condiciones de interconexión:

(HIba=HOb y HOa=HIab) implica que la paridad de HIba+HOa = LOb

y ya que el usuario de un nivel bajo conocen la paridad de LOa y LOb, entonces también conoce la paridad de HIa.
Si LOa y LOb son iguales entonces HIa es par, sin embargo un usuario de nivel bajo no conoce las entradas de nivel alto que han ocurrido en A.

4.6.1 Restricción

"Un sistema es restrictivo si los eventos fuera de la vista del usuario no afectan al conjunto de posibles secuencias de eventos de salidas en la vista del usuario" [24].

Es posible que una entrada de nivel alto no cambie el estado de nivel bajo del sistema [22].

4.6.1.1 Máquina de estado restrictiva

Una "máquina de estado de entrada total" puede aceptar una entrada de cualquier estado. Toda la información que puede ser obtenida en esta máquina proviene de las salidas ya que una entrada siempre es aceptada.

La información del estado del sistema y las secuencias de los eventos del sistema conforman lo que McCullough llama "vista del sistema del usuario".

Una vista de este tipo para un usuario de nivel bajo está completamente definida sobre las relaciones de equivalencia de los estados del sistema y sobre la secuencia de eventos. De esta manera dos estados son equivalentes si la única diferencia entre ellos está en la información de nivel alto. Dos secuencias de eventos son equivalentes si sus eventos de nivel bajo son iguales.

"Si dos secuencias de entradas T1 y T2 son aplicadas sobre dos estados equivalentes s1 y s2 respectivamente, esto produce dos nuevos estados equivalentes s1' y s2''' .

"Si dos estados equivalentes s1 y s2 producen dos salidas equivalentes T1 y T2 respectivamente, esto produce dos nuevos estados equivalentes s1' y s2' " .

4.7 REQUERIMIENTOS PARA LA INTERACCIÓN ENTRE DOMINIOS

A continuación se presenta una lista de elementos necesarios para la comunicación entre dominios diferentes, la cual es descrita en [2]. La lista no está completa, pero incluye un buen número de requerimientos, que podemos considerar como de los más comunes. Esta lista no sigue un orden específico.

1) Sintaxis de atributos y sintaxis interdominio [9,10].

2) Un lenguaje común para descripción de políticas [17,24].

3) Herramientas para la comparación de políticas [17,24,28].

4) Construcción bottom-up (construcción de políticas y modelos desde políticas y modelos previamente evaluados) [19,24,28].

5) Separación de políticas y mecanismos [14,19].

6) Integración de componentes de confianza [16].

7) Mecanismo para hacer cumplir la política en el origen [16].

8) Inclusión de políticas de usuario [18,20].

9) Inclusión de procesos de confianza [17].

10) Separación del modelo de seguridad de reglas de control acceso [17].

11) Combinación de políticas de seguridad [20,24].

12) Operaciones entre modelos (álgebra de modelos) [23].

13) Poder dividir el sistema en sus componentes [22].

14) Coordinación de interacción entre dominios y subdominios (política-subpolítica) [9,10,18,19].

15) Precedencia entre políticas [18,19].

16) Coordinación de interacción entre diferentes dominios (políticas) [9,10,19].

4.8 TIPOS DE INTERACCIÓN ENTRE DOMINIOS

En la figura 4.8.1 se presentan algunos de los tipos de interacción que se pueden presentar entre dominios. Estos tipos de interacción están numerados del I al V se representan con líneas que unen un dominio con otro. Las letras representan los diferentes sistemas.

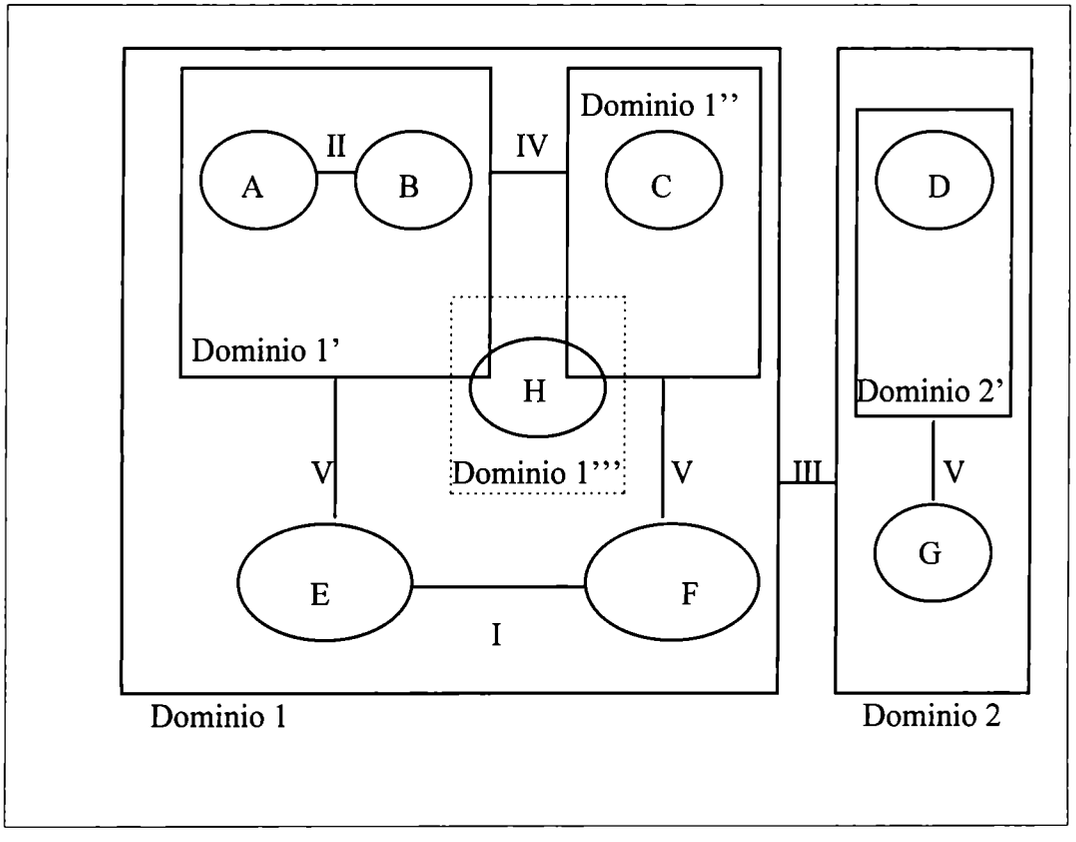

Fig. 4.8.1 tipos de interacción.

Para el tipo I, los sistemas E y F contienen la misma política, por lo que no es necesario una traducción de atributos o algún control adicional.

Lo mismo sucede para el tipo II, ya que los sistemas A y B poseen la misma subpolítica, en este caso el dominio 1' es subconjunto del dominio 1. Tampoco se necesita una traducción adicional de atributos o algún mecanismo de control extra.

En el tipo III, los sistemas interconectados pertenecen a dominios diferentes (1 y 2) por lo que es necesario una traducción de atributos y controles adicionales. La traducción de atributos se debe a que en el dominio 1 existe una política diferente a la del dominio 2, por lo cual los atributos de una no son válidos en la otra y viceversa. La traducción es un "mapeo" de atributos en el que se indica cuales atributos son equivalentes en ambos dominios.
En el tipo de interacción IV es necesario que se incluyan controles adicionales y traducción de atributos porque aunque los sistemas interconectados pertenecen al mismo dominio 1, también pertenecen a subdominios 1' y 1'' respectivamente. Es obvio que las políticas de los estos dominios son diferentes entre sí.

Para el tipo de interacción V, podemos observar que los sistemas pertenecen a un mismo dominio general, pero como en el caso anterior, estos sistemas también pertenecen a ciertos subdominios, lo cual hace que sea necesaria la presencia de controles adicionales y traducción de atributos.

4.8.1 Consideraciones

Algunas consideraciones que aparecen en [2] acerca de los requisitos para una comunicación entre dominios son las siguientes:

1) "Cuando las interacciones son consideradas en la etapa de concepción de la política".

2) "Cuando las interacciones serán implementadas en políticas de seguridad ya existentes".

3) "Cuando nuevas interacciones aparecen a causa de la evolución de las políticas de seguridad".

Los tipos de interacción no deben cambiar cuando evoluciona una política. Sin embargo, es posible que se adicionen o supriman algunos puntos de las políticas.

Al evolucionar las políticas, es posible que se incluyan características de otras políticas, pero no se deben traslapar. Como ejemplo de esto consideremos el sistema H en la figura 4.8.1 el cual contiene propiedades de los dominios 1,1',1" y 1'", pero no todas. Hay que notar que los tipos de interacción dependen de las características o propiedades de cada dominio, por ejemplo, para que el sistema H interactúe con los sistemas A, B o C no necesita ninguna traducción de atributos ya que el dominio al que pertenece contiene las propiedades necesarias para comunicarse sin problemas con dichos sistemas.

5 ENFOQUE DE CONTROL MANDATORIO

Actualmente, el problema de seguridad multidominio se trata de resolver utilizando una traducción o "mapeo" de los atributos de los datos para que estos sean intercambiados entre diferentes dominios. Este tipo de enfoques se han descrito anteriormente como el caso de Hosmer, BLP, McCullough, etc.
Sin embargo la traducción de atributos incrementa o degrada la información, haciendo difícil el conocer si todavía se preserva la seguridad. Esto se puede considerar como un enfoque de "Control Discresionario de Interacción".
La solución que se propone en el enfoque del presente trabajo es que se controlen en modo mandatorio todas las interacciones entre dominios diferentes (por ejemplo, donde no existe traducción de atributos) [1].

5.1 Descripción del enfoque

A continuación se abordará en detalle el trabajo de seguridad multidominio del Dr. Jesús Vázquez [1] el cual es el enfoque que se ha tomado para realizar el análisis e implementación de esta tesis.

En este enfoque, un conjunto de políticas de seguridad forman lo que se conoce como *dominio de seguridad* (ver fig. 5.1.1). Este conjunto de políticas forman la política de seguridad de ese dominio.

Cada dominio posee su propia política para controlar los accesos a los objetos. Un dominio puede consistir de uno o varios sistemas. Es recomendable que exista un mecanismo de autentificación para que todos los sujetos en el multidominio se autentifiquen de una manera uniforme (por ejemplo, Kerberos).

Los sujetos en un dominio pueden accesar objetos de otro dominio de acuerdo a las reglas del multidominio. Estas reglas forman la política multidominio o multipolítica. En este enfoque, al igual que los que se han visto hasta ahora, una multipolítica es el medio para controlar la interacción entre dominios de políticas diferentes. El requisito principal para llevar a cabo esta multipolítica es que los dominios fuente y destino posean algunas propiedades comunes de seguridad.
Cada propiedad de seguridad se asocia con un *nivel de abstracción*. Éste es un concepto que se maneja continuamente en este trabajo, y que se abordará en detalle más adelante. Los niveles de abstracción proveerán una jerarquía de propiedades en el multidominio la cual se llama *Jerarquía de Nivel de Abstracción* ALH (Abstraction Level Hierarchy).

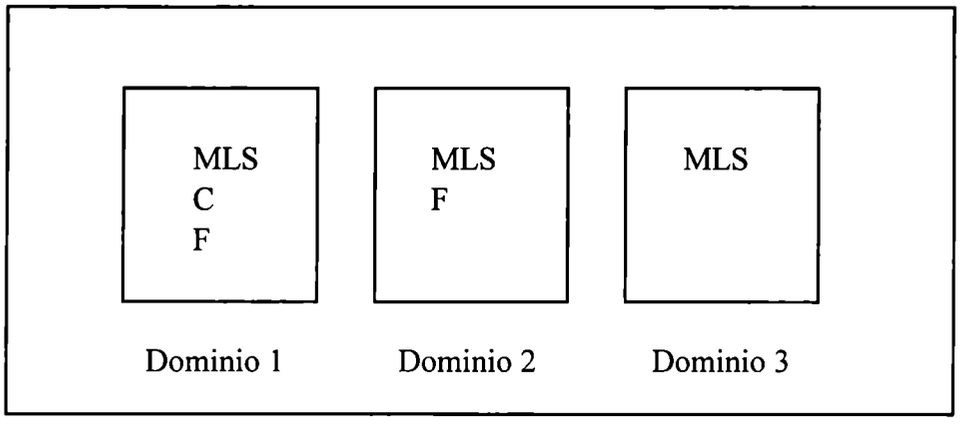

Fig. 5.1.1 Dominios de seguridad

Esta jerarquía se determina por medio de un criterio de intuición aplicado a cada propiedad de seguridad en el multidominio. Los niveles de abstracción ayudarán a definir la multipolítica de interacción.

Cada política que se contemple estará en un contexto multidominio y como consecuencia de ello, deberá interactuar en un entorno heterogéneo. Una política de seguridad en un multidominio es un conjunto de reglas tomadas de propiedades de políticas de seguridad diferentes, (por ejemplo, propiedad *, seguridad simple, conflicto de intereses, etc.). Una autoridad de dominio debe especificar su política escogiendo entre las propiedades de la ALH.

Los niveles de abstracción ayudan a decidir qué propiedades se deben incluir en cada paso de un proceso de especificación de políticas, además de que ayudan a la combinación y comprensión de propiedades.

La información multidominio de seguridad se almacena en una BD llamada Base de Información de Seguridad Multidominio (MDSIB por sus siglas en inglés). A esta BD la actualiza la Autoridad Multidominio, la cual contiene información general de seguridad. Por ejemplo:

- La Jerarquía del Nivel de Abstracción (ALH) del multidominio (que incluye todas las propiedades que pueden existir en las políticas de dominios).

- Los conjuntos de valores de atributos para todas las propiedades en la ALH. Una propiedad que es usada por dominios diferentes puede tener diferentes conjuntos de valores de atributos. La autoridad multidominio se encarga de colocarlos juntos de acuerdo a su estructura (Enrejado, subconjunto, etc.).

- La estructura de los valores de atributos para cada propiedad (cómo se relacionan los valores unos con otros, en ordenes parciales, relaciones de subconjuntos, etc.).

- Los dominios de seguridad actuales en el multidominio.

- El conjunto de propiedades y valores de atributos para cada dominio (esto constituye la política de dominio).

- El tipo de interacciones permitidas entre los dominios diferentes.

- La lista de dominios que se permite que interactuen.

5.2 Nivel de abstracción

En la actualidad, al comparar políticas de seguridad desde un mismo punto de vista, sólo se usa un nivel, ya que los sujetos y objetos son contemplados desde la misma perspectiva. Por el contrario, si se trata de combinar o comparar propiedades de dos puntos de vista diferentes en un único nivel, nos veremos inmersos en "problemas imposibles" [15]. En el enfoque que se implementa en el presente trabajo, se considera que las propiedades de seguridad no son comparables, pero si complementarias. De esta manera, se necesitan diferentes propiedades de políticas para modelar los accesos en el mundo real.

Las diferencias entre diversas políticas son comunes, por ejemplo, ciertos objetos para la política Financiera de Brewer y Nash [7] pueden ser vistos como *clases de conflicto de interés*, mientras que para la política comercial de Clark y Wilson [6] estos objetos pertenecen a grupos de *transacciones bien formadas*. Si tratamos de ubicar los objetos de este ejemplo en una política multinivel [4], entonces los objetos pertenecen a grupos de *clasificaciones*, por ejemplo, grupos de niveles de confidencialidad bajo condiciones jerárquicas sobre uso de objetos.

Debido a lo anterior, se dice que "Una Muralla China" no puede ser representada correctamente por el modelo de Bell y La Padula, o que el modelo "Lattice" no es suficiente para caracterizar políticas de "integridad", ya que la perspectiva de cada política es muy diferente.

5.3 Jerarquía de propiedades

Es útil asignar niveles a las diferentes propiedades de las políticas. Esto hace posible analizar la política general de seguridad de una forma natural.

Las propiedades para un entorno financiero no se diferencian entre los usuarios que ejecutan las operaciones, por ejemplo un empleado puede ejecutar las mismas operaciones que el director de la organización. Esto se puede evitar definiendo

transacciones bien formadas que establezcan las diferencias entre los roles de usuarios, esto visto desde la perspectiva comercial, donde exista una separación de funciones. Los procedimientos que modifican la información financiera de la dirección pueden ser ejecutados exclusivamente por el director de la organización.

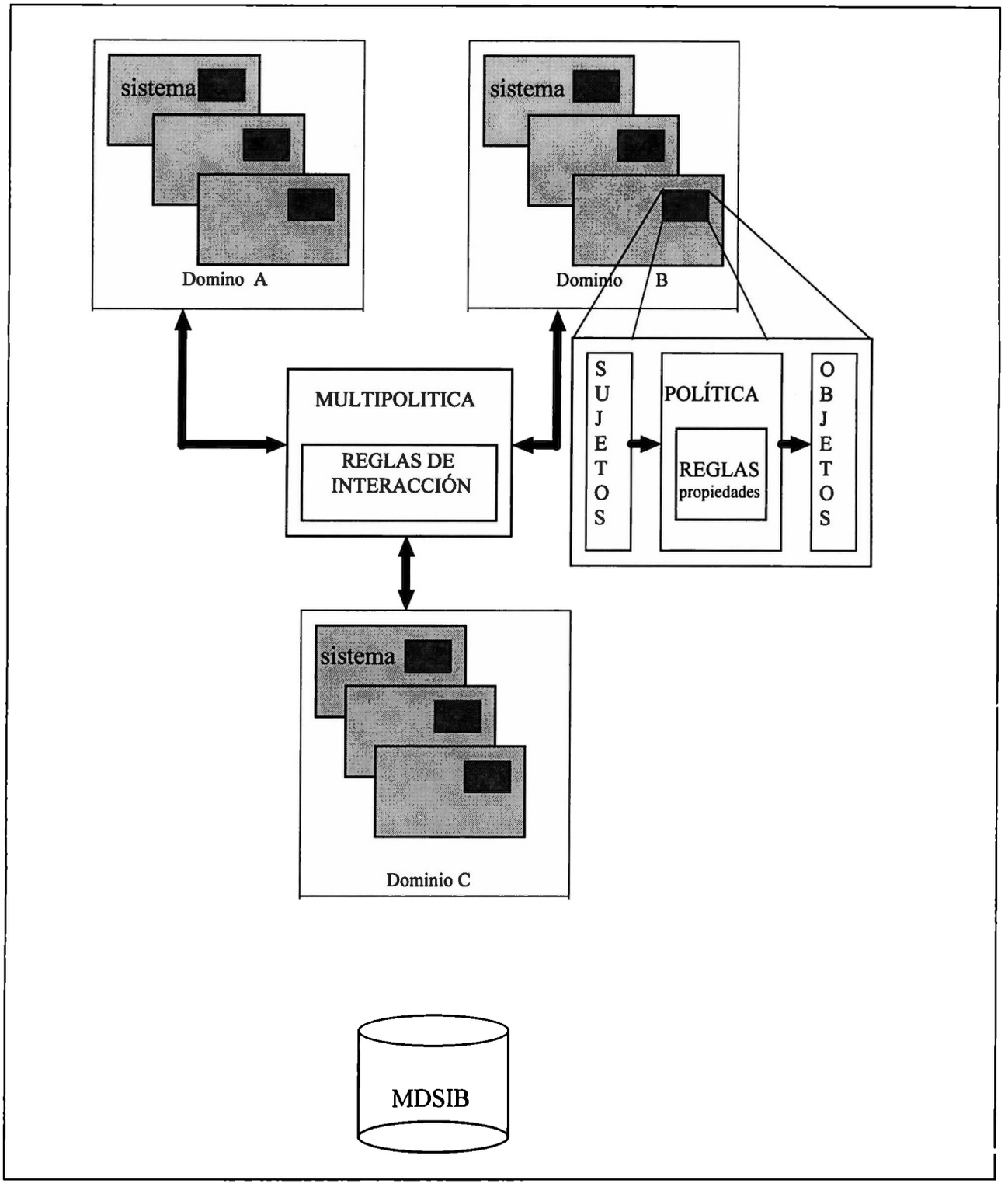

Fig. 5.3.1 Elementos del Multidominio

De manera similar a la anterior, desde un punto de vista de seguridad Multinivel, es necesario que todos los procedimientos que accesan a objetos (TP's) respeten las propiedades Multinivel en

toda transacción bien formada, de esta forma se media el acceso a la información considerando la naturaleza jerárquica de sujetos y objetos en la organización. Por ejemplo, el director de la organización está autorizado para accesar la información delicada a través de la ejecución de una transacción bien formada que es una operación financiera, respetando los posibles conflictos de intereses con otras organizaciones. En la figura 5.3.1 se muestran los elementos del Multidominio.

5.4 Determinación del nivel de abstracción

Los siguientes puntos son algunos criterios de intuición que permiten la determinación de los niveles de abstracción para propiedades de seguridad.

a. Dependencia: Cuando existen propiedades que se expresan en términos de otras propiedades, la propiedad dependiente tiene un nivel de abstracción mayor que la no dependiente. Estas propiedades ocupan niveles consecutivos en la jerarquía. Por ejemplo la propiedad de *separación de funciones* está incluida en la propiedad de *transacciones bien formadas*, por lo que la primera tiene un nivel mayor que la segunda.

b. Flujo de información: "Las propiedades que describen el flujo de información entre dos objetos tienen un nivel mayor que las propiedades que describen el acceso de un sujeto a un objeto. Cualquier flujo de información entre dos objetos puede ser visto como dos accesos, uno al objeto fuente y el otro al objeto destino". Pueden describirse flujos de información más complejos, por ejemplo, de un objeto a varios o de varios a uno, estando en un nivel de abstracción mayor.

c. Acceso dinámico: "Las propiedades del acceso dinámico tienen un nivel de abstracción más alto que las propiedades del acceso estático. La diferencia entre estos es que si las propiedades de acceso estático rechazan el acceso a un objeto, este rechazo siempre ocurrirá sin importar el acceso previo que el sujeto ha hecho. Por el contrario, las propiedades del acceso dinámico rechazan (o conceden) el acceso a un objeto dependiendo del acceso previo (incluyendo el acceso estático)".

5.5 Seguridad y vivacidad

Podemos dividir las propiedades de las políticas de seguridad en dos grandes tipos: Propiedades de seguridad (safety) y propiedades de vivacidad (liveness). La primera asegura que no pase algo malo, y la segunda que pase algo bueno eventualmente. En el enfoque se supone que las cosas malas son acciones que, si ocurren, comprometen la seguridad del sistema, y las cosas buenas son acciones que, si ocurren, no comprometen la seguridad del sistema o son buenas para él. Se propone usar propiedades de seguridad para describir restricciones sobre "cosas malas" y las propiedades de vivacidad para describir concesiones para "cosas buenas".

Un ejemplo de una propiedad de seguridad puede ser: "Un usuario no autorizado no puede accesar información delicada". En otras palabras, un usuario no autorizado "nunca" tendrá acceso a cierta información.

Una propiedad de vivacidad sería: "Un cajero lee la información del pago", por ejemplo, el usuario no lee esta información todo el tiempo, pero si a veces, cuando tiene que pagar.

Las fronteras referentes a la seguridad de sistemas son las siguientes:

- La frontera alta o "seguridad total" representa la propiedad de seguridad, donde ninguna acción puede pasar. En este caso toda acción dentro de un rango definido de acciones, es una cosa mala para el sistema.

- La frontera baja o "seguridad mínima" representa la propiedad de vivacidad, donde puede pasar cualquier acción. En este caso cualquier acción es considerada como una cosa buena para el sistema, es decir, no lo pone en riesgo alguno.

5.6 Jerarquía del nivel de abstracción

Las propiedades de las políticas del multidominio se deben ordenar por sus niveles de abstracción para formar una jerarquía. Como ejemplo se muestra la siguiente jerarquía en orden decreciente.

- Control de flujo de información no saneada
- Propiedad de opción libre
- Propiedad de conflicto de intereses
- Propiedad de separación de deberes
- Propiedad de transacción bien-formada
- Propiedad *
- Propiedad de seguridad simple

Se usó el criterio intuitivo descrito en la sección anterior para determinar este orden. Como podemos ver, también se establecen las siguientes dependencias "flujo de información no saneada" depende de la "propiedad de opción libre" y de la "propiedad de conflicto de interés", la "propiedad de opción libre" depende de la "propiedad de conflicto de interés", la "separación de deberes" depende de "transacciones bien formadas, la "propiedad *" depende de "seguridad simple". De acuerdo al criterio de flujo de información la "propiedad de transacción bien formada" es mayor que la "propiedad *" ya que puede ser considerada como un flujo de información complejo.
La propiedad de "control de flujo de información no saneada" es mayor que la propiedad de "selección libre" porque al llevar un control sobre la información sensible se tiene que controlar

de alguna manera la libre selección. En otras palabras, al hacer una selección no válida entonces se impedirá el flujo de la información.

Si aplicamos el criterio de seguridad-vivacidad, estas expresiones (propiedades) deben determinarse como algo que debe suceder o algo que no puede suceder, de esta manera dichas propiedades podrían se expresadas en un lenguaje formal como el de *lógica de seguridad* descrita en el capítulo anterior.

5.7 Multipolítica de interacción

Cuando dos dominios interactúan, la información que se recibe en el dominio destino debe poseer todos los atributos de seguridad de la política del dominio origen.
Esta información no necesariamente posee los valores adecuados para la política destino. Así, por ejemplo, la Autoridad Multidominio debe asignar los valores mínimos a los atributos en el dominio destino que no tengan su contraparte en el dominio origen. Por ejemplo, que todos los atributos no-comunes fueran inicializados a sus valores mínimos (ver la siguiente sección). Después de la verificación, se remueven los atributos no-comunes del dominio fuente. Entonces la multipolítica agrega a la información los atributos no comunes que llevan valores mínimos del dominio destino.

De esta manera la comunicación puede ser permitida sin importar que la política origen sea un subconjunto de la política destino.

5.8 Valores mínimos de atributos

Ya que el conjunto de valores de atributos asociados a una propiedad particular dependen de la autoridad del dominio que se ha seleccionado, estos valores pueden ser diferentes para la misma propiedad en diferentes dominios. Sin embargo existen valores que deben ser conocidos en todos los dominios, como por ejemplo los *valores mínimos*. Existe un valor mínimo para cualquier atributo de propiedad, (por ejemplo, el atributo *clearance* para seguridad simple debe tener el valor mínimo "no clasificado"; el atributo *clase de conflicto de intereses* para la política de la muralla China debe tener el valor mínimo "saneado"). La información que posee un valor mínimo en uno de sus atributos indica que no es una información sensitiva desde el punto de vista de la política asociada a dicho atributo.

5.9 Condiciones de la Multipolítica

Para permitir la interacción entre dominios se deben cumplir algunas condiciones. Estas condiciones son una parte de la descripción informal de la multipolítica que controla las interacciones en el multidominio.

- condición1. "La interacción entre dos dominios es posible si tienen al menos una propiedad de seguridad en común".

- condición2. "La interacción entre dos dominios posee comunicaciones seguras si sus conjuntos respectivos de propiedades contienen las *propiedades básicas* (consecuentemente se satisface la opción 1).
  Las propiedades básicas deben satisfacerse con cualquier política de seguridad en el multidominio ya que permite la comunicación segura y refleja la jerarquía implícita de los individuos en una organización. Se proponen las propiedades MLS como *propiedades básicas* (de acuerdo con la ALH propuesta).

- condición 3. Se aplica esta condición cuando se satisface la condición 2, en tal caso, el control de interacción deberá permitir la utilización del conjunto entero de valores de atributos asociados a cada una de las propiedades comunes (por ejemplo, todas las clasificaciones de seguridad para MLS, es decir; Super Secreto, Secreto, Clasificado, No-clasificado). Para las propiedades no-comunes, el control de interacción permitirá la utilización del conjunto respectivo de atributos pero con sus valores mínimos. Por ejemplo, se tienen dos dominios que tienen en común una propiedad comercial, una de ellas además tiene una propiedad financiera. Durante la interacción, las interacciones comerciales pueden usar como atributos valores "CDI's" (conjunto de datos restringidos) o "UDI's" (conjunto de datos no restringidos) mientras que las interacciones financieras sólo pueden usar "saneada" como valor de atributo (su valor mínimo), ya que si la información no es confidencial puede fluir libremente.

- condición 4. Se aplica esta condición cuando se satisface la condición 1 pero no la 2. En tal caso, el control de interacción permitirá el uso de un conjunto particular de valores máximos de atributos asociados a cada una de las propiedades comunes. Este conjunto de valores de atributos se determina por la autoridad multidominio (no deberá permitirse la información que posee valores altos de atributos debido a la ausencia de la seguridad de comunicación). Para las propiedades no-comunes, el control de interacción deberá permitir la utilización de los atributos respectivos a sus valores mínimos.

  Como un ejemplo de esta condición consideremos un dominio fuente donde exista una política multinivel y una comercial, y por otro lado, un dominio destino en el que exista una política multinivel únicamente. Supongamos que la información que se quiere transmitir es clasificada para MLS y Restringida para la política comercial (CDI). En este caso la condición 1 se cumple ya que en ambos dominios existe la política multinivel pero no se satisface la condición 2 porque la información es restringida.

- condición 5. Pueden interactuar dos dominios en un nivel de abstracción particular si sus políticas poseen un tipo de interacción de dominio-par. Las propiedades en niveles más altos deben considerarse como propiedades no-comunes.

5.10 Nivel de interacción

El nivel de interacción en una relación entre dos dominios se determina por la jerarquía de las propiedades de cada dominio.

Debe seleccionarse un nivel de interacción con relaciones de "Dominios Par", donde dos dominios tienen un subconjunto común de propiedades de cierta o ciertas políticas, incluyendo a las propiedades básicas.

Si los dominios interactuan en las propiedades básicas del nivel de abstracción, entonces el nivel de interacción es el nivel más bajo posible.

"Los dominios que poseen políticas descritas por los mismos niveles jerárquicos de abstracción pueden seleccionar el nivel e interacción de esta jerarquía (ver condición 5 de la sección anterior)".

Para una relación de dominio par entre el Dominio 1 y el Dominio 2, se puede usar cualquier nivel de abstracción para interactuar.

Tabla 5.10.1 Ejemplo 1 de niveles de interacción

DOMINIO 1	DOMINIO 2
propiedad n	propiedad n
...	...
propiedad y	propiedad y
propiedad x	propiedad x
...	...
prop. básicas	prop. básicas

Si uno de los dominios en una relación, posee las propiedades del otro (relación dominio a subdominio), además de las propias, y la condición 2 se cumple, entonces el nivel más alto de interacción será igual al nivel más alto de abstracción en común, (por ejemplo, el nivel de abstracción de la propiedad *y* que se muestra a continuación).

Tabla 5.10.2 Ejemplo 2 de niveles de interacción

DOMINIO 1	DOMINIO 2
propiedad n	---
...	...
propiedad z	---
propiedad y	propiedad y
propiedad x	propiedad x
...	...
prop. básicas	prop. básicas

En este ejemplo se puede observar que el nivel más alto de interacción es el de la propiedad ***y*** en ambos dominios.

Si dos dominios interactuantes poseen al menos una propiedad en común (nivel de abstracción) y no se satisfacen las relaciones del dominio par y la del dominio al subdominio, entonces es una relación dominio a dominio. La interacción debe cumplir con la propiedad 4.

Tabla 5.10.3 Ejemplo 3 de niveles de interacción

DOMINIO 1	DOMINIO 2
propiedad n	propiedad n
propiedad n-1	propiedad w
...	...
propiedad z	---
propiedad y	propiedad y
propiedad x	propiedad x
...	...
prop. básicas	propiedad a

En el ejemplo de la tabla 5.10.3, se puede ver que la propiedad en común es ***n*** y coincide con el nivel más alto de abstracción.

5.11 Propiedades básicas

En todas las organizaciones existe una connotación implícita de jerarquías. Un ejemplo de jerarquía en una organización podría ser: presidente, directores, jefes, personal administrativo, personal de ventas, etc. Esta jerarquía debe describirse mediante una estructura de niveles como se hace con las clasificaciones en una política multinivel.
Las operaciones que manejan información en favor de los usuarios de la organización deben realizarse considerando su estructura multinivel. Además, los sistemas distribuidos necesitan controles, tales como control de acceso y el control de flujo de información entre los dispositivos de red. Estos controles deben aplicarse cuando los sujetos y objetos se clasifican en forma diferente o poseen atributos diferentes. Estos controles requieren:

* Usar utilerías comunes de sistema (funciones de correo, servicios de directorio, servidores de BD).

* Accesar objetos remotos pasando sobre varios dispositivos de red.

Dichos controles deben reforzarse usando propiedades de una política, (por ejemplo, un MLS aplicado a sistemas distribuidos).
Si consideramos que las propiedades de la política MLS (seguridad simple y seguridad *) pueden ser fundamentales en una relación entre dominios, entonces a estas propiedades se les puede considerar como básicas. Como puede inferirse, estas propiedades básicas constituyen el nivel de abstracción más bajo en la ALH.

6 IMPLEMENTACION DE LA AUTORIDAD MULTIDOMINIO

En este capítulo se expone los detalles del desarrollo de la Autoridad Multidominio, la cual es la parte fundamental del sistema Multidominio. Se tratan aspectos como el de la comunicación con otros procesos del sistema como *enviar* y *servidor*. El programa *enviar* es el que utilizan los usuarios para solicitar un envío de información y *servidor* es el proceso que se encuentra en el destino y se encarga de realizar la evaluación de la política local y de recibir los mensajes en caso de que la comunicación sea aprobada.

6.1 Función de la Autoridad Multidominio

La función de la Autoridad Multidominio (AM) es la de controlar el intercambio de información entre diversas políticas de seguridad computacional. Este control se lleva a cabo mediante una implementación cliente-servidor, donde éste último es la AM. También se incluye en el sistema Multidominio un servidor de mensajes, quien se encarga de la recepción de mensajes después de la comunicación ha sido autorizada.

Los clientes poseen políticas locales de seguridad. Se escogió este enfoque centralizado por simplicidad. Una posible extensión sería contar con una AM distribuida en el multidominio. El servidor AM recibe las peticiones y crea un proceso para atender a cada cliente.

Para implementar el sistema de interacción multidominio se seleccionaron tres políticas: Multinivel (MLS), Comercial (C) y Financiera (F) [3]. La razón para haber seleccionado estas políticas es porque son de las más estudiadas.

Esta implementación tiene como propósito probar experimentalmente el enfoque propuesto en [1].

Para el desarrollo de este enfoque se presupone que se está interactuando con un sistema operativo y canales de comunicación seguros, es decir, nadie puede modificar los atributos de la información que circula en el multidominio.

Se desarrolló una política que controla la interacción entre diferentes políticas residentes en 30 máquinas.

La implementación del sistema se realizó en el Laboratorio de Cómputo Especializado del ITESM-CEM. El lenguaje que se utilizó fue C. La comunicación se realizó por medio de sockets[2].

[2] herramienta de comunicación en unix

Las políticas locales que se implementaron en cada máquina del Laboratorio de Cómputo Especializado tienen por objeto evaluar el intercambio de información entre dichas máquinas, con la AM controlando la interacción. Los detalles de la elaboración de las políticas locales se encuentra en el trabajo de tesis de Maestría, en proceso, realizado por Norma Angélica Ramírez y resultan de las diferentes combinaciones de las tres políticas tradicionales mencionadas anteriormente.

Las entidades que interactúan pueden tener cualquier política local y será la Autoridad Multidominio (AM) la que determine si se puede establecer la comunicación. Esta decisión se toma siguiendo los criterios adecuados para cada política, es decir, para la política MLS, por ejemplo, se deberá verificar, en primer lugar, que exista tanto en el origen como en el destino. MLS es la política que se toma como propiedad básica para el desarrollo de este enfoque, por lo que siempre debe estar presente en los dominios de seguridad que deseen comunicarse bajo dicho enfoque. Si se quisiera establecer una interacción con otro dominio pero MLS no estuviera presente en el origen o en el destino, la AM simplemente impide dicha interacción.

	Financiera	Nivel 3
	Comercial	Nivel 2
Propiedad Básica	Multinivel	Nivel 1

Fig. 6.1.1 MLS, política fundamental

La figura 6.1.1 muestra las políticas y sus niveles. MLS es indispensable para este enfoque.

Las políticas Financiera y Comercial pueden omitirse tanto en el origen como en el destino. Esta omisión puede realizarse porque las políticas C y F no forman parte de las propiedades básicas. Por ejemplo, en caso de que C se encuentre presente en el origen pero no en el destino, se deberá cumplir la condición de que la información sea no restringida. De esta manera la información pasará del origen al destino, sin que se corra algún riesgo, ya que puede ser conocida por todo el mundo.

En el caso contrario al ejemplo anterior, es decir, que C se encuentre en el destino pero no en el origen, la AM le asignará un atributo *UDI* a la información para ser enviada. Cuando se recibe la información, la política C verificará que el atributo es para información no restringida y permitirá la interacción.

Con lo anterior podemos observar y deducir que cuando la información enviada tiene atributos para C y/o F, y en el destino no existen estas políticas, es responsabilidad de la AM evaluar si se puede establecer la comunicación. Esta comunicación será posible si los atributos son los más bajos, es decir, "datos no restringidos" para C y "saneados" para F.

Si la AM autoriza una comunicación entre dominios, el host destino recibirá un datagrama que debe llevar los campos para los tres atributos de las políticas MLS, C y F independientemente de la combinación de estas tres políticas que resida en el host destino. Aquí se puede ver que la AM asigna los atributos mínimos mencionados anteriormente, ya que el destino tiene que recibir un valor para el campo-atributo de cada política.

Después de que la interacción es permitida por la AM, la responsabilidad será de la política local. Ésta evaluará los atributos de la información y determinará si es aceptada o no de acuerdo a los atributos del destino.
En la figura 6.1.2, la AM ha autorizado la interacción entre el origen y destino. En caso de que no se autorice, la AM enviaría un mensaje al origen en el que se le indica que no puede establecer comunicación con el destino. Cuando esto ocurre el programa que realiza la petición imprime un mensaje en la pantalla y termina.

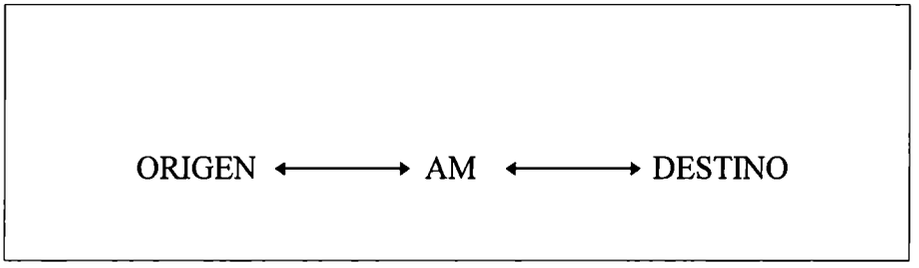

Fig. 6.1.2 Flujo de datos en la autorización.

En los siguientes párrafos se mencionan los detalles del diseño y construcción de la Multipolítica. La autoridad multidominio (AM) controla una tabla donde están registrados los host y las políticas que poseen. Los campos de esta tabla tienen la forma de la figura 6.1.3:

```
struct politicas
      {
       char host[12];
       int  mls[5];
       int  com[5];
       int  fin[5];
      }
```

Fig. 6.1.3 estructura de políticas por host

La declaración del identificador con que nos referiremos a la tabla es un arreglo. Por ejemplo:
struct politicas pol[30];

La tabla 6.1.1 muestra un ejemplo de uso datos asociados en esta estructura.

Tabla 6.1.1 Políticas por host

host	mls	com	fin
jamaica	1	1	1
surinam	1	1	1
guatemala	1	1	0
panama	1	0	0
...	...		...

El valor de cada uno de estos campos será igual a 1 si la política existe en el host. Si no existe la política, entonces el valor del campo entero será igual a 0.

En la tabla 6.1.1 es donde la AM verifica las políticas del host destino. Para los host jamaica y surinam podemos observar que su política es una combinación de las tres que se manejan en el presente trabajo, mientras que guatemala tiene una combinación de Multinivel y Comercial, y panama tiene únicamente la Multinivel, que es lo mínimo permitido.

Si una de las políticas individuales del emisor no existiera en el destino y la información que se envía es sensible, simplemente, la AM prohibe la comunicación ya que el destino no puede interpretar los datos recibidos. Este es un tipo de control mandatorio propuesto en [1] a diferencia del enfoque de Hosmer que propone un control discresionario.
Con este tipo de control se puede observar que no existe la traducción de atributos, con lo cual se impide caer en el problema de asignar un valor a un atributo que no existe en la política del destino.

El proceso de transferencia de información se hace a través de un cliente y el servidor. Éste último es precisamente la AM. Por su parte, el cliente utiliza un comando para hacer el envío de datos. Este comando tiene la forma siguiente:

enviar <us-origen us-destino MLS C F>

donde:
us-origen = usuario origen
us-destino = usuario destino
MLS = atributo de la información para la política Multinivel

C = atributo de la información para la política Comercial
F = atributo de la información para la política Financiera

El comando *enviar* establece una conexión, por medio de sockets, del usuario origen con la AM. Los parámetros MLS, C, F y us-destino son enviados por el origen y recibidos por la AM.
Este comando obtiene el identificador de host en el que se está ejecutando y también es enviado a la AM. De la misma forma obtiene los valores para los atributos de las políticas MLS, C y F de las tablas de host y usuarios.

Con estos datos la AM determina si puede haber una interacción entre el usuario que está haciendo la petición y el destino requerido por dicho usuario. Para realizar esto, la AM verifica en su tabla de políticas cual es la política del host origen y la del destino.

Una vez que la Multipolítica ha autorizado la interacción, transfiere el datagrama al servidor de mensajes. Este último aplicará la política local y determinará si la comunicación es aprobada desde el punto de vista de su dominio. La respuesta de la política local será

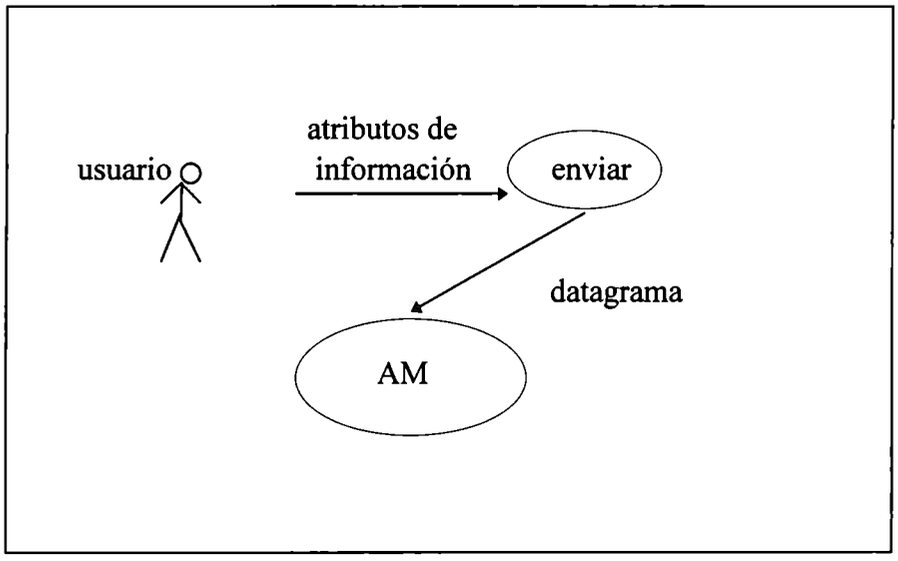

Fig. 6.1.4. enviar obtiene valores de atributos y transfiere a la AM.

enviada a la AM desde el servidor de mensajes. La AM a su vez, enviará esta respuesta al cliente que solicitó la comunicación. Si la respuesta que reciba el cliente es positiva, entonces podrá enviar mensajes al destino solicitado. El servidor de mensajes está ejecutándose en *background,* en cada una de las máquinas que se utilizaron para la implementación. La petición al servidor de mensajes se realiza desde el cliente de la AM, es decir, desde el programa *enviar.* La comunicación entre el cliente y la AM se realiza usando un sólo canal para el envío y recepción de datos en ambos procesos. Lo mismo sucede para la comunicación entre la AM y el servidor de mensajes, es decir, se usa el mismo canal para el envío del datagrama de la AM al servidor de

mensajes y para la recepción de la respuesta de evaluación de la política local (contenida en el servidor de mensajes).
La información que recibe el destino es la que envía la AM en un datagrama con todos los datos necesarios para que la política local evalúe si la comunicación es factible.
La información enviada al destino es una estructura como la que aparece en la figura 6.1.5.

```
struct datag
   {
    int  pol_o_mls;
    int  pol_d_mls;
    int  pol_o_com;
    int  pol_d_com;
    int  pol_o_fin;
    int  pol_d_fin;
    char us_origen[10];
    char us_destino[10];
    char host_origen[12];
    char host_destino[12];
    char mls[10];
    char com[10];
    char fin[10];
    char cci[10];
    char cdc[10];
    char us_o_cat_mls[3];
    char us_d_cat_mls[3];
    char us_o_clas_mls[4];
    char us_d_clas_mls[4];
    char us_o_cci[10];
    char us_d_cci[10];
    char us_o_cdc[10];
    char us_d_cdc[10];
    char us_o_objeto[10];
    char us_d_objeto[10];
    char us_o_tp1[12];
    char us_d_tp1[12];
    char us_o_tp2[12];
    char us_d_tp2[12];
   }data;
```

Fig. 6.1.5 Datagrama utilizado en la interacción

Cuando la AM y la política local han aprobado la comunicación, se establece una conexión entre el cliente y el servidor de mensajes.

La descripción de cada uno de los campos del datagrama usado en la comunicación es la siguiente:

- pol_o_mls. Si su valor es 1 indica que la política MLS existe en el origen.
- pol_d_mls. Si su valor es 1 indica que la política MLS existe en el destino.
- pol_o_com. Si su valor es 1 indica que la política C existe en el origen.
- pol_d_com. Si su valor es 1 indica que la política C existe en el destino.
- pol_o_fin. Si su valor es 1 indica que la política F existe en el origen.
- pol_d_fin. Si su valor es 1 indica que la política F existe en el destino.
- us_origen. Nombre del usuario origen.
- us_destino. Nombre del usuario destino.
- host_origen. Dirección del host origen.
- host_destino. Dirección del host destino.
- mls. Atributo de la información para la política MLS.
- com. Atributo de la información para la política Comercial.
- fin. Atributo de la información para la política Financiera
- cci. Clase de Conflicto de Interés para la política Financiera.
- cdc. Conjunto de Datos de Compañía para la política Financiera.
- us_o_cat_mls. Categoría del usuario origen en MLS
- us_d_cat_mls. Categoría del usuario destino para MLS.
- us_o_clas_mls. Clasificación del usuario origen en MLS.
- us_d_clas_mls. Clasificación del usuario destino en MLS.
- us_o_cci. Clase de Conflicto de Interés del usuario origen.
- us_d_cci. Clase de Conflicto de Interés del usuario destino.
- us_o_cdc. Conjunto de Datos de Compañía del usuario origen.
- us_d_cdc. Conjunto de Datos de Compañía del usuario destino.
- us_o_objeto. Objeto asociado al usuario origen.
- us_d_objeto. Objeto asociado al usuario destino.
- us_o_tp1. TP asociado al usuario origen.
- us_d_tp1. TP asociado al usuario destino.
- us_o_tp2. TP asociado al usuario origen.
- us_d_tp2. TP asociado al usuario destino.

6.1.1 Forma de evaluación de la Autoridad Multidominio

La multipolítica revisa en primer lugar los atributos de MLS para el origen y el destino, en segundo lugar los atributos para C y por último los atributos para F.
En este caso **no se aplica la jerarquía que se emplea en las políticas locales fin(com(mls())) porque no es necesario, ya que implicaría trabajo extra el hecho de que se evalúe MLS en segundo o tercer lugar si esta política no existiera en algún dominio.**

Los reportes de las evaluaciones con interacciones entre diferentes dominios de seguridad se generan automáticamente por la AM. Esto no sólo sirve como un fin de la tesis, sino que es una herramienta de auditoría necesaria en al AM.

6.2 ANÁLISIS DE LA AUTORIDAD MULTIDOMINIO

La Autoridad Multidominio debe controlar la interacción entre dominios diferentes, que son combinaciones de las políticas Financiera, Comercial y Multinivel. En la presente sección se considerará que una interacción es permitida desde el punto de vista de la AM si ésta envía la información al destino después de hacer el análisis respectivo (aplicar la Multipolítica). En otras palabras, si al llegar la información al host destino, proveniente de la AM, la política local no la acepta, esto no es competencia de la AM.

El nivel de interacción entre dos dominios está determinado por las políticas (MLS, C, F) que componen la política del origen y la del destino.
Por ejemplo, si dos dominios interactúan únicamente con MLS entonces su nivel de interacción es el más bajo. Mientras que dos dominios que interactúen con MLS, C y F tendrán el nivel de interacción más alto.

Si la comunicación entre dos dominios es permitida y su nivel de interacción es el más alto (incluyendo las tres políticas) entonces la comunicación entre sus subdominios es permitida. Por ejemplo, si ambos dominios tienen las tres políticas, pueden comunicarse únicamente usando la política MLS.
Cabe notar que se podrían haber utilizado más políticas pero para los propósitos de la tesis, con estas políticas es suficiente.

La interacción entre dos dominios es "completa" si ambos dominios contienen exactamente la misma política.

Sea D_1 y D_2 dos dominios y $D_1 \subset D_2$ y $D_2 \subset D_1 \Rightarrow$ la interacción es completa.

- El acceso a cualquier dominio es garantizado $\Leftrightarrow P_{1o}(h_o,b) = P_{1d}(h_d,b)$ and $P_{2o}(h_o,b) = P_{2d}(h_d,b)$ and ... and $P_{no}(h_o,b) = P_{nd}(h_d,b)$ (interacción completa)

donde:
n = número de política.
P_{no} = política n en el origen.
h_o = host origen.
b = valor booleano. Es igual a 1 si la política n existe en el host indicado por h. Es igual a 0 si la política n no existe en el host indicado por h.

P_{nd} = política n en el destino.
h_d = host destino.

Ejemplos de políticas:
$P_{1o}(1,1)$=MLS

La política 1 en el host origen (1) es MLS.

$P_{1d}(2,1)$=MLS

La política 1 en el host destino (2) es MLS.

M es el Multidominio (conjunto de todas las políticas)
Sea D_1 y D_2 dominios
Si $D_1 \subset D_2 \Rightarrow$ la interacción es no-completa

Una comunicación entre dos dominios, donde uno es subconjunto de otro mantiene una interacción no-completa.

En este caso el nivel de interacción entre D_1 y D_2 < nivel mayor de interacción de D_2.
El máximo nivel de interacción de un dominio está dado por el número de políticas que lo componen.

La comunicación siempre es garantizada si se cumple que $A_o(P) = 0 \; \forall \; P$

donde $A_o(P)$ = atributo de la política P en el origen.

Para las políticas consideradas en el presente trabajo tenemos que:

u = 0	para MLS	(u = unclassified)
udi = 0	para C	(udi=unconstrained data item)
s = 0	para F	(sanitized)

6.2.1 Acceso fuertemente restrictivo

Como ya se mencionó, el enfoque de [1] no permite traducción de atributos de políticas. La Autoridad Multidominio permite la interacción entre dos dominios D_1 y D_2 para una política P si :

$D_1 = D_2$ o
$D_1 \subset D_2$ o
$D_2 \subset D_1$
Un caso particular es el siguiente:

$D_1 \not\subset D_2$ y $D_2 \not\subset D_1$ y $A_o(P) = 0$

Aquí se puede ver que D_1 y D_2 son disjuntos. En este caso no puede suceder la comunicación debido a que en el enfoque de [1] se manejan propiedades básicas o valores mínimos. La propiedad básica, es decir, la que está contenida en todos los dominios que pueden enviar y recibir mensajes es la política Multinivel.

Considerando el caso anterior, donde los dominios son disjuntos, es decir, no contienen ninguna política en común, la siguiente condición no sería válida para establecer el enlace:

$D_1 \not\subset D_2$ y $D_2 \not\subset D_1$ y $P_{1o}(h_o,0)$ y $P_{1d}(h_d,0)$ y $P_{2o}(h_o,0)$ y $P_{2d}(h_d,0)$ y ... y $P_{no}(h_o,0)$ y $P_{nd}(h_d,0)$ y $A_o(P) = 0$

donde P=MLS.

La razón por la que no es permitida la comunicación es la ausencia de la política MLS, ya que es el valor mínimo.

En general la condición mínima para que la comunicación entre dos dominios D_1 y D_2 se establezca es:

- ($D_1 = D_2$ and $A_o(mls) <= A_d(mls)$) or $A_o(mls)=0$ o
- ($D_1 \subset D_2$ and $A_o(mls) <= A_d(mls)$) or $A_o(mls)=0$ o
- ($D_2 \subset D_1$ and $A_o(mls) <= A_d(mls)$) or $A_o(mls)=0$ o

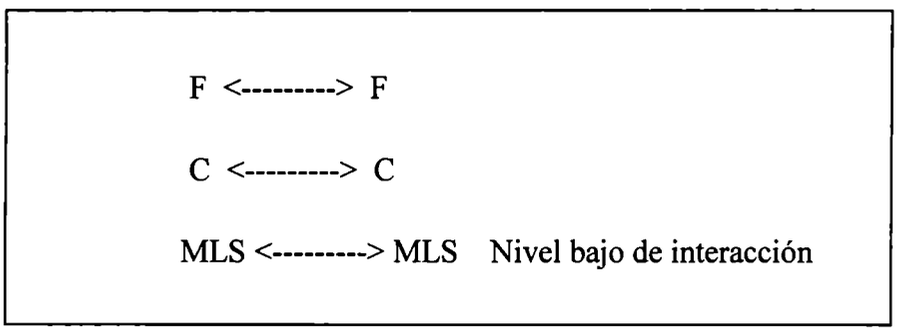

Fig. 6.2.1.1 MLS como interacción mínima

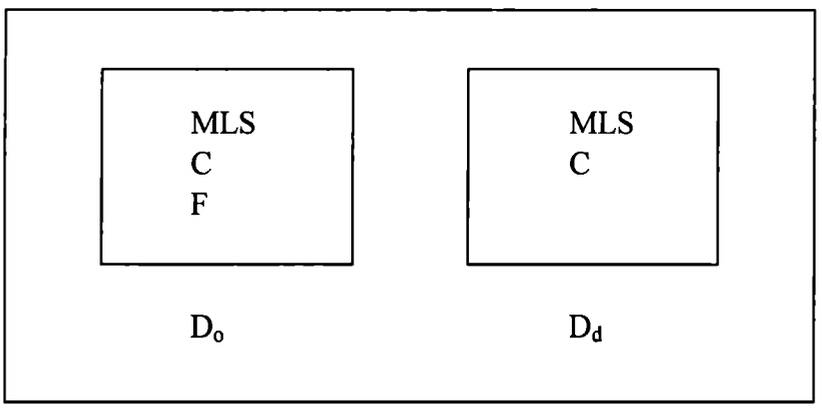

Fig. 6.2.1.2 Interacción permitida para información no sensible en D_o

Para dos dominios, D_1 y D_2, la mínima interacción que pueden establecer está dada por su nivel más bajo de interacción.
El nivel más bajo de interacción está determinado por las propiedades básicas. Para este enfoque el nivel más bajo de interacción es, como ya se ha mencionado a nivel de la política, MLS, sin importar las combinaciones de políticas MLS, C y F.

Si existe una comunicación entre un dominio origen D_o y uno destino D_d y hay una interacción no completa y $D_d \subset D_o$ entonces **existe información no sensible en el origen**.

7 ADMINISTRACIÓN DEL SISTEMA

Para realizar las evaluaciones, la AM debe conocer las propiedades de los usuarios y las políticas en cada dominio. Las bases de información de seguridad multidominio (BISMD) incluyen los datos de usuarios, y las políticas asociadas a cada host. Esto último representa un dominio en particular.

Para poder manipular estas propiedades y políticas se han implantado programas que permiten ingresar nuevos usuarios con sus respectivos atributos, así como nuevos hosts con sus políticas. También permiten modificaciones, eliminaciones y listado de datos ya existentes. En general, los programas de administración actúan sobre : usuarios, hosts y puertos. Con el archivo de puertos, la AM selecciona el puerto por el que se realizará la conexión con la máquina destino.
Los datos de usuarios, hosts y puertos están contenidos en archivos específicos con extensión *adm*. El archivo de usuarios tiene el nombre de *atrib.adm,* el de políticas por host es *politica.adm* y el de puertos es *puertos.adm*.
Para los usuarios, el archivo contiene una tabla con todos los datos que requiere la AM para realizar la evaluación. Estos datos son:

- Identificador del usuario.- Es el nombre del usuario dentro del sistema.
- Host al que pertenece el usuario.- Es la máquina en la que reside el usuario.
- Clasificación MLS.- Es la clasificación del usuario en la política MLS.
- Categoría MLS .- Es la categoría de la información en la política MLS.
- Objeto.- Indica la entidad que podrá manipular el usuario en la política C.
- Procedimiento de transformación 1.- Programas a los que tiene derecho el usuario.
- Procedimiento de transformación 2.- Programas a los que tiene derecho el usuario.
- Clase de conflicto de interés.- Es sector al que pertenece el usuario.
- Conjunto de datos de compañía.- Son los datos de una compañía específica a los que el usuario tiene derecho de accesar.

La AM debe leer estos valores del archivo adecuado y colocarlos en el datagrama que es enviado a la máquina destino.

La política local verifica los datos de usuario en el archivo correspondiente y determina si la comunicación puede realizarse entre el origen y el destino. Por ejemplo; si el usuario que solicita la transferencia de mensajes tiene una clasificación "t" (top secret) y la clasificación de la información que desea enviar es "s" (secret), la política local (en el destino) verifica que el usuario destino posea la clasificación "s" o superior. Si es así, este nivel de interacción es aprobado. El usuario origen puede enviar y recibir información con cualquier tipo de clasificación, ya que "t" es la más alta para MLS.

La clasificación "s" del usuario destino quiere decir que éste puede recibir información *secreta.* Lo anterior especifica que se realizan dos evaluaciones, una por parte de la AM y la segunda por parte de la política local.

Una situación similar se presenta con los otros niveles de interacción. Por ejemplo, para un usuario que desee enviar información y que pertenezca a una clase de conflicto de interés "X" y a un conjunto de datos de compañía "Y", se autorizará la interacción si el usuario destino pertenece a un mismo conjunto de datos de compañía o bien, a una clase de conflicto de interés diferente. En otras palabras, para el usuario destino debe cumplirse que

$$cci \neq \text{"X"} \quad or \quad cdc = \text{"Y"}$$

Un ejemplo de la estructura del archivo de usuarios aparece en la tabla 7.1.

Tabla 7.1 Atributos de usuarios

Usuario	Host	Clas	Cat	Objeto	TP1	TP2	CCI	CDC
al310444	mexico	t	t	mensaje	enviar	servidor	educación	ITESM
al310445	guatemala	s	s	mensaje	enviar	servidor	financiera	AMEXCO
al310446	guatemala	t	t	mensaje	enviar	servidor	salud	IMSS
al310478	barbados	t	t	0	0	0	educación	UNAM
al310482	guatemala	c	c	0	0	0	0	0
...	...	...						...

En la tabla 7.1 podemos observar que el usuario al310444 pertenece al host *mexico,* tiene una clasificación "t" para MLS, el objeto asociado al usuario es *mensaje* y los procedimientos que permiten su envío y recepción están presentes, el conjunto de datos de compañía es *ITESM* y su clase de conflicto de interés es *educación.* Los ceros para los atributos de usuario significan que en el dominio de ese usuario no está contenida la política correspondiente. Esto debe coincidir con el archivo de hosts, que es el que contiene las políticas por cada máquina.

Para la política Comercial se ha tomado únicamente el objeto *mensaje* porque es el único que se necesita en el sistema. En general, cualquier objeto que el usuario pueda accesar tendrá que estar en la lista de atributos de usuario. Es natural que si el sistema utilizara una lista grande de objetos, entonces los atributos de usuario para la política C deberán estar en un archivo diferente. Estos atributos, sin embargo, se han incluido en el archivo *atrib.adm* por simplicidad.
La política local accesa *atrib.adm* cuando requiere evaluar la política Comercial. Lo primero que hace la AM es verificar que C exista en el origen y el destino. Si el dominio origen y el destino

contienen esta política, la política local debe verificar los objetos y sus TP's asociados para determinar si la interacción en este nivel es permitida.

Para el caso del ejemplo de la tabla 7.1, si el usuario al310444 quisiera enviar un mensaje al usuario al310445, la política local permitiría esta interacción (para la política C). La razón por la que se permite esta interacción es porque el usuario origen tiene un TP1=enviar para el objeto *mensaje*, es decir, puede ejecutar este programa. Además el usuario destino tiene un TP2=servidor para el objeto mensaje, lo que quiere decir que este usuario puede recibir mensajes. El hecho de que el usuario destino tuviera un TP1=0 y TP2=servidor para el objeto *mensaje* no afecta la recepción de mensajes, pero si el envío, ya que no podrá realizarlo.
Es importante hacer notar que al310478 y al310482 no pueden enviar mensajes a ningún usuario ya que no tienen el objeto necesario para realizar esta acción. Aunque tuvieran el objeto asociado, *si no* poseen el TP adecuado no podrán enviar o recibir mensajes.

7.1 Administración de usuarios

Es posible realizar inserciones, eliminaciones, cambios o consultas del archivo de atributos de usuarios.
Para efectuar una inserción de un usuario nuevo, se deberá usar el comando *atrins*. El formato de este comando es el siguiente:

atrins <INTRO>

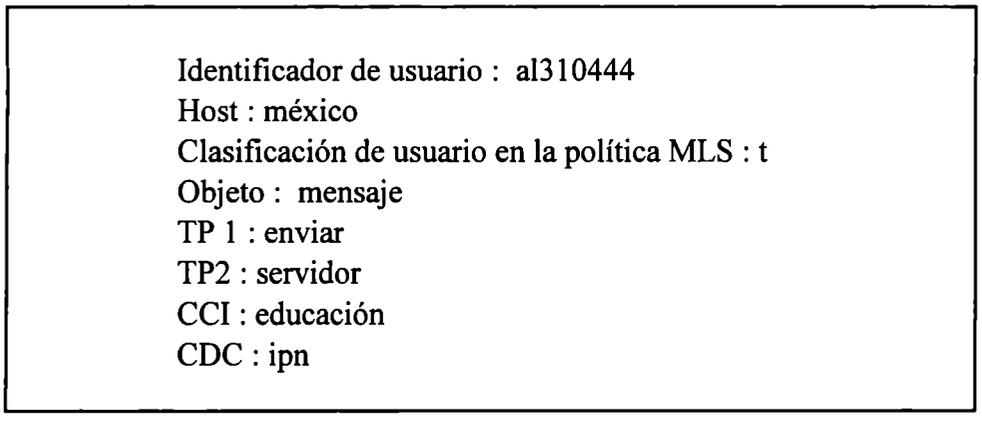

Fig. 7.1.1 Inserción de usuario nuevo.

Los datos que pedirá el programa después de que se ha tecleado el comando para inserción de atributos son : identificador de usuario, el host en el que se encuentra el usuario, la clasificación del usuario en la política MLS, objeto que se manejará en la política C, procedimientos de transformación para la política C, clase de conflicto de interés para la política F y conjunto de datos de compañía también para F.

El programa para insertar atributos de usuarios está validado de tal manera que sólo permite dar de alta a usuarios nuevos en el sistema. Si se intenta dar de alta un usuario que ya existe en el archivo *atrib.adm*, el programa simplemente desplegará un mensaje indicando que ese usuario ya está en el sistema. Para lograr esta validación el programa de inserción de usuarios realiza una búsqueda del usuario que se ha ingresado. Esta búsqueda es similar a la que se realiza con el programa de consultas de usuarios.
Para realizar una consulta de atributos de usuarios se debe usar el comando *atrcon*. El formato de este comando es:

atrcon <usuario>

```
if ((atrib = fopen("atrib.adm", "r"))== NULL)
  {
   fprintf(stderr, "Imposible abrir archivo.\n");
   return 1;
  }

 fseek(atrib, 0, 0);
 fread(us_id, sizeof(us_id), 1, atrib);
 while( strcmp(argv[1],us_id) && ! feof(atrib) )
  {
    posicion=ftell(atrib);
   fseek(atrib,posicion+86,0);
   fread(us_id, sizeof(us_id), 1, atrib);
   if ( !strcmp(us_id,argv[1] ) )
 found=1;
 }

 if ( ! found )
 {
  printf("\n usuario %s no existe \n", argv[1] );
  fclose(atrib);
  exit(1);
 }
if ( found )
{
 posicion=ftell(atrib);
 fseek(atrib,posicion,0);
 fread(us_host, sizeof(us_host), 1, atrib);
 . . .
```

Fig. 7.1.2 Consulta de usuarios.

donde usuario, es el identificador del usuario. La ejecución de este comando dará como resultado una lista de atributos de usuario para las tres políticas consideradas en este trabajo. Por ejemplo:

atrcon al310444 <INTRO>

El resultado de esta consulta sería similar a los datos presentados en la tabla 7.1.1.
Al utilizar este comando, en realidad se está consultando el archivo *atrib.adm*.
Una parte del programa que realiza la consulta de datos aparece en la figura 7.1.2.

La figura 7.1.2 muestra la localización del primer usuario en el archivo de usuarios *atrib.adm*. Posteriormente aparece el ciclo "while" donde se realiza la evaluación de la condición para localizar al usuario pasado como parámetro al programa de consultas. En otras palabras, se compara el usuario leído desde el archivo de usuarios con el parámetro recibido por el programa. Esta comparación termina hasta que se encuentre el identificador de usuario buscado o bien, hasta que se encuentre el fin de archivo.

Cuando un usuario ha sido localizado en el archivo *atrib.adm*, se despliegan los atributos asociados a éste. En la figura 7.1.2 se muestra el código para la impresión del primer atributo de usuario. Esta figura contiene puntos suspensivos al final, porque se ha omitido la impresión de los demás atributos de usuario, ya que esto, se realiza de manera similar en todos los casos.
Cuando un usuario no ha sido localizado el sistema despliega un mensaje que indica esta situación.

Otro de los programas que componen el sistema es el que realiza modificaciones a los atributos de usuarios. El formato del comando usado para este fin es :

atrcam <usuario> u || h || m || c || f

Este programa realiza cambios con los nuevos valores asignados por el usuario. Para modificar el identificador de usuario se debe utilizar *u*. Si se incluye el parámetro *h* se podrá realizar un cambio en el host del usuario. Los parámetros *m, c* y *f* sirven para realizar una modificación en los atributos del usuario en la política MLS, C y F respectivamente. En cada cambio sólo se puede incluir un parámetro.

Un ejemplo de una actualización es el siguiente:

atrcam al310444 h <INTRO>

Los valores de los atributos del usuario al310444 cambiarán por los que se ingresarán con la petición de datos por el programa *atrcam*. Si no se desea cambiar algún o algunos valores de atributos, se deberá presionar <ENTER>. Por ejemplo :

Host (méxico) : <INTRO>

no modificará el valor de este campo.
El programa ignora las actualizaciones cuando detecta un <ENTER> en un campo de atributo. El 0 en algún atributo indica que el usuario ya no podrá enviar mensajes.

```
if(!strcmp(argv[2],"u"))
 {
    printf("\n nuevo identificador -> ");
    gets(us_id_nuevo);
    fseek(atrib, 0, 0);
    fread(us_id, sizeof(us_id_nuevo), 1, atrib);
    while( strcmp(us_id_nuevo,us_id) && ! feof(atrib) )
     {
      posicion=ftell(atrib);
      fseek(atrib,posicion+86,0);
      fread(us_id, sizeof(us_id_nuevo), 1, atrib);
      if ( !strcmp(us_id_nuevo,us_id) )
              found=1;
     }
    if( found )
     {
      printf("\n identificador de usuario ya existe ");
      fclose(atrib);
      exit(1);
     }

    if (strcmp(us_id,""))
     {
      posicion=ftell(atrib);
      fseek(atrib,posicion-10,0);
      fwrite(us_id, sizeof(us_id), 1, atrib);
     }
    printf(" usuario: %s",us_id);
    }

   if(!strcmp(argv[2],"h"))
   {
    printf("\n nuevo host -> ");
    gets(us_host);
    if (strcmp(us_host,""))
    {
     posicion=ftell(atrib);
     fseek(atrib, posicion, 0);
     fwrite(us_host, sizeof(us_host), 1, atrib);
    }
 }
```

Fig. 7.1.3 Modificaciones de atributos de usuario

Para realizar la actualización, primero se localiza el usuario indicado y se posiciona el apuntador del archivo de usuarios en su primer atributo. Cuando se intenta actualizar un usuario que no se encuentra en el sistema se desplegará un mensaje indicando que dicho usuario no existe. Un segmento del programa para cambios de usuarios se presenta en la figura 7.1.3.

Otra función de la administración de usuarios es la eliminación de usuarios. Para realizar esto, se tiene que hacer uso del comando *atreli*. Se requiere sólo un parámetro, que es el identificador de usuario. El programa de eliminación de usuarios realiza una búsqueda para verificar que el usuario pasado como parámetro exista. En caso de que la búsqueda sea exitosa, el sistema pregunta si se desea borrar ese usuario, a lo cual se debe responder con una *N* o *S*. El uso del comando es de la siguiente forma:

atreli <usuario>

usuario es el identificador del usuario. El sistema realiza un borrado físico de los datos, para ello utiliza un archivo temporal *temp.adm* en el que se guardan los datos de los usuarios del sistema que sean diferentes del usuario pasado como parámetro. El siguiente paso es guardar el archivo temporal con nombre *atrib.adm*.

La última función de la administración en lo correspondiente a usuarios es el listado de usuarios del sistema. Esto se realiza con el comando *atrlist*. En la tabla 7.1.1 aparece un ejemplo del listado de usuarios.

Tabla 7.1.1 Listado de usuarios del sistema

usuario	host
al310444	mexico
al310445	guatemala
al310446	guatemala
al310447	nicaragua
....	...

7.2 Administración de dominios

Para la administración de cada uno de los *host* se utilizan cinco programas; uno para la inserción de nuevas estaciones de trabajo con sus respectivas políticas (hostins), el segundo es para la consulta de políticas por host (hostcon), el tercero es para modificaciones de atributos de estaciones de trabajo (hostcam), otro es para dar de baja estaciones de trabajo (hosteli) y el último es para listar los hosts que existen en el sistema (hostlist).

La administración de estaciones de trabajo también se puede ver como la administración de las políticas de los diferentes dominios. Cada host tiene asociadas una o más políticas de las tres que se han propuesto en este trabajo, por lo tanto cada host representa un *Dominio de Seguridad.*

Para inserción de nuevas estaciones de trabajo en el sistema, es necesario usar el comando:

hostins <host m || c|| f || -m || -c || -f >

Este comando lo que hace es insertar en el archivo de estaciones de trabajo un nuevo host con sus políticas asociadas.

- host.- Es el nombre del host que se desea incluir en el sistema.
- m.- Es un atributo para la política MLS que indica que ésta será incluida en el host.
- c.- Es un atributo para la política C que indica que ésta será incluida en el host.
- f.- Es un atributo para la política F que indica que ésta será incluida en el host.
- -m.- Es un atributo para la política MLS que indica que ésta no será incluida en el host.
- -c.- Es un atributo para la política C que indica que ésta no será incluida en el host.
- -f.- Es un atributo para la política F que indica que ésta no será incluida en el host.

En el archivo *politica.adm*, los datos para cada política por host serán guardados como ceros o unos.
El número 1 significa que existe esta política en el host referido. Por el contrario, los ceros indican que esa política no existe en el host.

Por ejemplo:

hostins jamaica m c

En este ejemplo se está insertando un host nuevo en el sistema. El identificador del host es *jamaica* y contendrá las políticas Multinivel y Comercial.

Un fragmento del código que realiza la inserción aparece en la figura 7.2.1.

No es necesario incluir los tres parámetros para políticas en el comando *hostins,* se puede incluir uno como mínimo y tres como máximo.
Este programa de inserción, contempla la negación de escritura en el archivo de estaciones de trabajo cuando éstas están repetidas. Si un usuario intenta dar de alta un host que ya existe en el sistema, el programa desplegará un mensaje "host ya existe" y terminará.
Para la consulta de políticas por host se utiliza el comando *hostcon.* La forma de usarlo es:

hostcon <host>

Como se puede ver en este ejemplo, sólo se requiere el nombre del host como parámetro. El resultado de una ejecución como

hostcon jamaica <INTRO>

```
strcpy(h_pol.host,argv[1]);
strcpy(h_pol.mls,argv[2]);
strcpy(h_pol.com,argv[3]);
strcpy(h_pol.fin,argv[4]);
fwrite(h_pol.host, sizeof(h_pol.host), 1, hosts);
fwrite(h_pol.mls, sizeof(h_pol.mls), 1, hosts);
fwrite(h_pol.com, sizeof(h_pol.com), 1, hosts);
fwrite(h_pol.fin, sizeof(h_pol.fin), 1, hosts);
```

Fig. 7.2.1 Escritura de nuevo host con sus políticas

serán las políticas que contiene dicha estación de trabajo, por ejemplo;

mls = 1
com = 1
fin = 0

El argumento pasado a este programa en la función *main* es el nombre del host. En el fragmento del programa aparecen las asignaciones de los argumentos a los campos de la estructura *h_pol*, los cuales serán guardados en el archivo *politica.adm*.

El archivo de *politica.adm* se abre en modo "lectura" únicamente, ya que se trata de una consulta. La búsqueda del host se realiza con el ciclo "while" siguiente:

```
while( strcmp(argv[1],h_pol.host) && ! feof(hosts) )
```

donde se compara el argumento pasado a la función principal con el primer identificador de host del archivo *politica.adm*. Si no son iguales estos dos valores y no se ha encontrado el fin del archivo entonces se toma el siguiente identificador de host hasta que se encuentre el nombre correcto o el fin de archivo.

Cuando se encuentra el host correcto, se toman los valores de las políticas asociadas a dicho host y se imprimen en pantalla.

```
if (argc!= 2)
 {
  printf("\numero incorrecto de parametros ");
 exit(1);
 }

 if ((hosts = fopen("politica.adm", "r+"))== NULL)
 {
  fprintf(stderr, "Imposible abrir archivo.\n");
  return 1;
 }
 fseek(hosts, 0, 0);
 fread(h_pol.host, sizeof(h_pol.host), 1, hosts);
 posicion=ftell(hosts);
 fseek(hosts, posicion, 0);
 while( strcmp(argv[1],h_pol.host) && ! feof(hosts) )
 {
  posicion=ftell(hosts);
  fseek(hosts,posicion+6,0);
  fread(h_pol.host, sizeof(h_pol.host), 1, hosts);
  if (!strcmp(argv[1],h_pol.host) )
        found=1;
 }
if ( !found )
 {
  printf("\n host %s no existe \n", argv[1]);
  fclose(hosts);
  exit(!);
}
 if ( found )
  {
  printf("\n host: %s",h_pol.host);
  posicion=ftell(hosts);
  fseek(hosts,posicion,0);
  fread(h_pol.mls, sizeof(h_pol.mls), 1, hosts);
  printf("\n mls: %s",h_pol.mls);
  . . .
  fclose(hosts);
  return 0;
  }
```

Fig. 7.2.2 Consulta de políticas por host

Como se mencionó anteriormente, el sistema Multidominio también contempla la modificación del archivo de estaciones de trabajo.

El programa que realiza esta función es *hostcam*. El formato del comando para actualizaciones es el siguiente:

hostcam <host > m || -m || c || -c || f || -f

donde *host* es el nombre de la estación de trabajo que se desea actualizar, *m* es la política Multinivel, *c* es la Comercial y *f* es la Financiera. Los parámetros con guión indican que la política será suprimida del dominio especificado en el primer argumento. El programa colocará un 1 o un 0, dependiendo del parámetro, en el campo que corresponde a la *política* en la base de datos designada para guardar la información de los Dominios (*política.adm*). Suponga que existe un host llamado *surinam* que contiene la política Multinivel y la Comercial y se desea incluir la Financiera y suprimir la Comercial. Entonces se tendrá que utilizar el comando *hostcam* de la siguiente manera:

hostcam surinam -c f <INTRO>

Después de la actualización indicada con el ejemplo anterior, cambiará el atributo de la política F para el host *surinam*. El valor para la política MLS de *surinam* permanece sin cambio, por lo que los valores de atributos serán:

mls = 1
com = 0
fin = 1

El código de la figura 7.2.3 muestra la parte donde se realiza la selección de una acción, dependiendo del, o de los parámetros pasados al programa. Este fragmento de código del programa *hostcam* sólo incluye la verificación de los parámetros recibidos por el programa para la política MLS, el código completo se muestra en el anexo B.
Es importante destacar que no se puede cambiar el nombre del host y si fuera necesario cambiarlo, se tiene que borrar y posteriormente insertar el nuevo nombre de host con sus políticas. La validación de los valores recibidos en los parámetros del programa *hostcam* se pueden observar en la figura 7.2.4. Sólo se muestra la parte de la validación para el caso de que se reciban cinco parámetros. Para el caso de tres y cuatro parámetros el código es similar. Por ejemplo, para tres parámetros recibidos, sólo se incluye la comparación donde aparece argv[2], para cuatro parámetros se incluyen las comparaciones donde aparece argv[2] y argv[3].

Cuando el programa recibe tres parámetros se ha solicitado sólo un cambio, argv[0] es el nombre del programa, argv[1] es el nombre del host y argv[2] es el indicador de la política que cambiará.

```
switch(argc)
 {
 case 3:
    if( !strcmp(argv[2],"m") )
     {
       incluye_mls();
       break;
     }
    if( !strcmp(argv[2],"-m"))
     {
       elimina_mls();
       break;
     }

 case 4:
    if( !strcmp(argv[2],"m") || !strcmp(argv[3],"m") )
       incluye_mls();
    if( !strcmp(argv[2],"-m") || !strcmp(argv[3],"-m") )
       elimina_mls();

 case 5:
  if( !strcmp(argv[2],"m") || !strcmp(argv[3],"m") || !strcmp(argv[4],"m") )
       incluye_mls();
  if( !strcmp(argv[2],"-m") || !strcmp(argv[3],"-m") || !strcmp(argv[4],"-m") )
       elimina_mls();

}
```

Fig. 7.2.3 Selección para cambiar atributos de host

El orden para los argumentos de políticas de este comando no es importante, o sea que, se pueden colocar de cualquier forma y los cambios se llevarán a cabo. Para el ejemplo anterior se pudo haber escrito :

hostcam surinam f -c <INTRO>

y el resultado sería exactamente el mismo.

Después de la validación mostrada en la figura 7.2.4, se realiza la búsqueda del host que se ha especificado con el comando *hostcam*. Si no se encuentra el host, el sistema lo indicará por medio de la impresión de un mensaje en pantalla.

La figura 7.2.5 muestra las funciones utilizadas por el programa *hostcam* para realizar modificaciones en la política Multinivel de un dominio.

```
int main(int argc, char *argv[])
{
  parametros=argc;
  if(argc<3 && argc>5)
     {
      printf("\n numero incorrecto de parametros\n");
      exit(1);
     }
  switch (argc)
  {
   case 3:
      strcpy(mls,argv[2]);
   case 4:
      strcpy(com,argv[3]);
   case 5:
      strcpy(fin,argv[4]);
  }
if(argc==5)
 {
 if( strcmp(argv[2],"m") && strcmp(argv[2],"c") && strcmp(argv[2],"f") )
 if( strcmp(argv[2],"-m") && strcmp(argv[2],"-c") && strcmp(argv[2],"-f") )
  {
    printf("\n parametro %s no valido \n",argv[2]);
    exit(1);
  }
 if( strcmp(argv[3],"m") && strcmp(argv[3],"c") && strcmp(argv[3],"f") )
 if( strcmp(argv[3],"-m") && strcmp(argv[3],"-c") && strcmp(argv[3],"-f") )
  {
    printf("\n parametro %s no valido \n",argv[2]);
    exit(1);
  }
 if( strcmp(argv[4],"m") && strcmp(argv[4],"c") && strcmp(argv[4],"f") )
 if( strcmp(argv[4],"-m") && strcmp(argv[4],"-c") && strcmp(argv[4],"-f") )
   {
    printf("\n parametro %s no valido \n",argv[2]);
    exit(1);
   }
 }
 ...
```

Fig. 7.2.4 Validación de argumentos recibidos por *hostcam*

Para eliminar hosts del sistema se utiliza el comando *hosteli*. La forma de emplearlo es la siguiente :

hosteli <host>

donde *host* es el identificador de la máquina que se desea eliminar.

```
int incluye_mls()
{
  strcpy(mls,"1");
  fseek(ap,pos_act,0);
  fwrite(mls, sizeof(mls),1,ap);
  return 1;
}

int elimina_mls()
{
  strcpy(mls,"0");
  fseek(ap,pos_act,0);
  fwrite(mls, sizeof(mls),1,ap);
  return 1;
}
```

Fig. 7.2.5 Funciones de modificación para MLS

El sistema realiza una baja física, utilizando un proceso similar al de eliminaciones de atributos de usuario.
Por último, la orden *hostlist* muestra la lista de máquinas dentro del sistema. Su formato es simplemente

hostlist <INTRO>

El resultado de la ejecución de este comando aparece en la figura 7.2.6.

```
mexico
guatemala
nicaragua
salvador
panama
colombia
dominicana
brasil
argentina
...
```

Fig. 7.2.6 Lista de hosts

7.3 Selección de puertos

Cuando la AM recibe una petición de servicio debe analizar el dominio destino al cual se quiere comunicar el cliente. Esto es un requisito fundamental ya que la AM debe enviar la información necesaria al servidor local para que éste evalúe la posibilidad de comunicación de acuerdo a su política. Como se mencionó al principio de este trabajo, la comunicación entre los procesos que forman el sistema es por medio de sockets. Para establecer la comunicación se requiere que el proceso que hace la petición indique el número de puerto del servidor. Para que la AM conozca el número de puerto asociado al destino elegido por el cliente que hace la solicitud de envío de mensajes, tiene que accesar esta información de un archivo llamado *puertos.adm*. Este archivo forma parte de la *información de seguridad del multidominio*. El archivo de puertos contiene información como la mostrada en la tabla 7.3.1.

Tabla 7.3.1 Puertos asociados a cada host

host	puerto
jamaica	3100
barbados	3200
mexico	3300
guyana	3400
surinam	3500
belice	3600
ecuador	3700
...	...

La Autoridad Multidominio localiza el host que el cliente le ha enviado como destino y toma su número de puerto. El puerto encontrado por la AM se utiliza para realizar la conexión como se muestra en la figura 7.3.1.

```
portnumber = puerto_t;

if ((outfd = u_connect(portnumber, data.host_destino)) < 0)
 {
  u_error("Incapaz de establecer una conexión Internet");
  exit(1);
 }
```

Fig. 7.3.1 Conexión al puerto del host destino.

En la figura 7.3.1 aparece una variable llamada *puerto_t* que es donde la AM asigna el número de puerto leído del archivo de puertos. Este valor, a su vez, es asignado a la variable *portnumber* que es la que se utiliza como parámetro de la función *u_connect* para llevar a cabo la conexión.

Cuando la AM recibe una petición de comunicación por parte de un cliente, tiene que conocer cual es el puerto asociado al host destino. Para realizar esto, la AM utiliza un código como el de la figura 7.3.2. En este fragmento del programa de la AM se observa la parte en que se realiza la búsqueda del puerto. Esta búsqueda se lleva a cabo de manera similar a las búsquedas en los programas de consultas para usuarios y máquinas vistos anteriormente. Después de este fragmento de programa se utiliza una función *fread* para leer el número de puerto que pertenece al host localizado. La AM busca la máquina especificada como *host_destino* en el datagrama enviado por el programa cliente.

```
fseek(ap, 0, 0);
fread(host, sizeof(host), 1, ap);
posicion=ftell(ap);
fseek(ap, posicion, 0);
while( strcmp(argv[1],host) && ! feof(ap) )
{
 posicion=ftell(ap);
 fseek(ap,posicion+12,0);
 fread(host, sizeof(host), 1, ap);
}

if ( strcmp(host,argv[1]) )
 {
  printf("\n host %s no existe",argv[1]);
  exit(1);
 }
```

Fig. 7.3.2 Localización de un puerto

Para la administración de los puertos se han implementado programas como en los casos anteriores para atributos de usuarios y políticas por host.

El primer programa que analizaremos es el que permite dar de alta nuevas máquinas con un número de puerto asociado.

El nombre del programa para realizar esta tarea es *ptoins*. El formato del comando es el siguiente:

ptoins <host num-pto>

donde host es el identificador de la máquina y num-pto es el número de puerto asociado a ella. Suponga que se desea insertar el host *guyana* con un número de puerto *5000*, entonces se deberá usar :

ptoins guyana 5000 <INTRO>

El programa para ingresos de puertos efectúa una búsqueda del host especificado como argumento y si no lo encuentra en el archivo de puertos entonces procede a darlo de alta. En caso de que el host ya exista en el sistema sólo se despliega un mensaje indicando esta situación y termina el programa. Lo mismo sucede para un número de puerto que ya exista en el sistema, es decir que ya se encuentre asociado a un host. Una parte del programa se presenta en la figura 7.3.3.

```
fseek(ap, 0, 0);
fread(host, sizeof(host), 1, ap);
posicion=ftell(ap);
fseek(ap, posicion, 0);
while( strcmp(argv[1],host) && ! feof(ap) )
{
 posicion=ftell(ap);
 fseek(ap,posicion+12,0);
 fread(host, sizeof(host), 1, ap);
}
if ( !strcmp(host,argv[1]) )  {
  printf("\n host %s ya existe",host);
  exit(1);
 }
fseek(ap, 0, 0);
fread(puerto, sizeof(puerto), 1, ap);
posicion=ftell(ap);
fseek(ap, posicion+12, 0);
while( strcmp(argv[1],puerto) && ! feof(ap) )
{
 posicion=ftell(ap);
 fseek(ap,posicion+12,0);
 fread(puerto, sizeof(puerto), 1, ap);
}
```

Fig. 7.3.3 Código para ingresar un host con un puerto asociado

En el código de la figura 7.3.3 incluye la búsqueda del host. Si en esta búsqueda no se localiza el host, entonces se procede a realizar la búsqueda del número de puerto. Si no se localiza el puerto, se procede a la escritura en el archivo de puertos del nuevo host y su puerto.

Para realizar consultas de los puertos en el sistema asociados a los hosts, el sistema cuenta con el comando *ptocon.* El formato de este comando es :

ptocon <hosts>

donde host es el identificador de la máquina. El resultado de la ejecución de este comando es número de puerto asociado a la máquina que se ha pasado como parámetro.
Para realizar cambios en los puertos del sistema, se utiliza el programa *ptocam.* El formato de uso es :

ptocam <host> <p || h>

donde *host* es el identificados de la máquina, *p* es el parámetro para indicar un cambio en el puerto y *h* para el identificador del host.

Para eliminar un puerto asociado a una máquina, el sistema cuenta con el programa *hosteli.* Este programa utiliza sólo el nombre del host como parámetro, con el cual localiza la información que será borrada. La eliminación es de manera física y funciona de manera similar a las eliminaciones de atributos de usuario y hosts. El formato de comando es :

ptoteli <host>

El último programa para la administración de puertos, es el que lista los hosts existentes en el sistema con sus correspondientes puertos. El formato de esta instrucción es :

ptolist <INTRO>

El resultado de la ejecución anterior es similar a la figura 7.3.1.

7.4 ATRIBUTOS DE LA INFORMACIÓN

Para hacer una petición de envío de mensajes, el usuario tiene que emplear el comando *enviar*. El formato de este comando es:

enviar <us_origen us_destino m c f >

Un ejemplo del uso de este comando es el siguiente:

enviar al310445 al310449 t udi s

lo que significa que el usuario al310445 desea enviar mensajes al usuario al310449 con los atributos *t* para la política MLS, *udi* para C y *s* para F.

Los programas para el cliente, la AM y el servidor de mensajes utilizan la misma estructura para almacenar datos. El programa cliente lee del archivo *atrib.adm* los atributos de usuario y los coloca en el datagrama. Posteriormente envía los datos a la AM y ésta, a su vez, envía la misma estructura al servidor de mensajes. Los atributos que usa el cliente, es decir, los atributos de la información, se transmiten como parte de la estructura.
Los atributos del comando *enviar* no tienen efecto alguno en la AM si las políticas en el origen y destino son idénticas, aún cuando la información que se desea transmitir sea sensible. Esto se debe a que la AM solo verificará que las políticas existan tanto en origen como en el destino. En este caso, la negación o aprobación de la transmisión corresponden a la política local, que se encuentra implantada en el servidor de mensajes.

En el caso de que el dominio origen posea una combinación de políticas mayor que el destino, la AM deberá verificar que la información posea un atributo *no confidencial* para aquellas políticas que no tengan su contraparte en el destino.

En el caso contrario, cuando el destino posea una combinación de políticas más grande que el origen, la AM deberá poner el atributo *no confidencial* en el campo correspondiente del datagrama.

7.5 IMPLEMENTACIÓN DE LA COMUNICACIÓN

La comunicación entre los usuarios que desean realizar un envío de mensajes está desarrollada con sockets en un diseño cliente-servidor. El usuario debe emplear el comando *enviar* con sus respectivos parámetros. Para utilizar un número de puerto que permita establecer la conexión por medio de sockets con el servidor, se utiliza una tabla de puertos de hosts. Esta tabla contiene un puerto asociado a cada máquina de la red, de esta manera se puede tener un servidor local en cada host recibiendo peticiones en un número de puerto fijo. Más adelante se tratará con detalle como utiliza la AM a la tabla de puertos. El servidor local está bloqueado esperando alguna petición por parte de los clientes.
Cuando la AM recibe una petición de servicio, crea un proceso hijo y éste atiende a dicha petición. De esta manera el servidor puede atender a un todos los clientes en la red.
La parte de código del servidor que realiza esta tarea está en la figura 7.5.1.

La función *atiende(communfd)* está incluida en el código del hijo. Esta función es llamada siempre que un cliente hace una solicitud. Esta función realiza la lectura del datagrama que ha sido enviado por el programa *enviar*. Para ello utiliza la sentencia

```
while ((bytesread=u_read(communfd, data, sizeof(data)) > 0)
```

donde *bytesread* es una variable de tipo entero, *u_read* es una función que está contenida en el archivo *uici.h*, data es el datagrama y por último *sizeof(data)* es el tamaño del datagrama que se recibe. Después de recibir los datos, la AM realiza la evaluación y determina si la comunicación se puede establecer.

```
portnumber = (u_port_t)atoi(argv[1]);
  if ((listenfd = u_open(portnumber)) == -1)
  {
   u_error("No se pudo establecer una conexion de puerto");
   exit(1);
  }
  while ((communfd = u_listen(listenfd, client)) != -1)
  {
   fprintf(stderr, "[%ld]: Una conexion fue establecida con %s \n",
              (long) getpid(), client);
   no_client ++;
   if ((child = fork()) == -1)
   {
     fprintf(stderr, "No se pudo crear un hijo \n");
     break;
   }

   if (child == 0)
   {                    /* Codigo del hijo */
     u_close(listenfd);
     fprintf(stderr, "\n \tProceso [%ld] atendiendo la conexion con %s (%d) \n",
              (long) getpid(), client, no_client);
     atiende(communfd);
     u_close(communfd);
     fprintf(stderr, "\n \tProceso [%ld] ha finalizado de atender a: %s (%d) \n",
              (long) getpid(), client, no_client);
     exit(1);
   }
```

Fig. 7.5.1 Fragmento del código de la AM

En caso de que la transmisión sea negada por la AM, ésta enviará un mensaje, donde se indique dicha negación, al cliente que hizo la solicitud y la ejecución del comando *enviar* terminará. En el caso de que la interacción sea autorizada desde el punto de vista de la AM, ésta envía el datagrama al servidor de mensajes. El servidor de mensajes estará ejecutándose en el host destino y permanecerá bloqueado hasta que reciba una petición. El servidor de mensajes se puede implementar de dos formas: la primera, que es creando un proceso hijo por cada petición que recibe y la segunda es recibiendo todas las solicitudes en una estructura *for* o *while* y mostrar los mensajes en el orden correcto. La primera función que realiza el servidor de mensajes es la evaluación de la política local. Una vez realizada esta función, transmite un mensaje a la AM de aprobación o negación de la interacción desde el punto de vista del dominio local. La AM, a su

vez, transmite al cliente la respuesta, y si ésta es afirmativa, el cliente podrá transmitir mensajes al destino elegido.

```
while( (bytesread = u_read(communfd, buf, BLKSIZE)) > 1)
 {
byteswritten = write(STDOUT_FILENO, buf, bytesread);
if (bytesread != byteswritten)
  {
    fprintf(stderr, "Error escribiendo %d bytes, %ld bytes escritos\n",
        (long)bytesread, (long)byteswritten);
    break;
  }
```

Fig. 7.5.2 Código de AM para recepción mensajes provenientes del cliente

La parte de la recepción de mensajes, en la AM, provenientes del cliente se puede ver en la figura 7.5.2. Esta parte se ejecuta si las condiciones del enfoque de interacción han sido satisfechas.

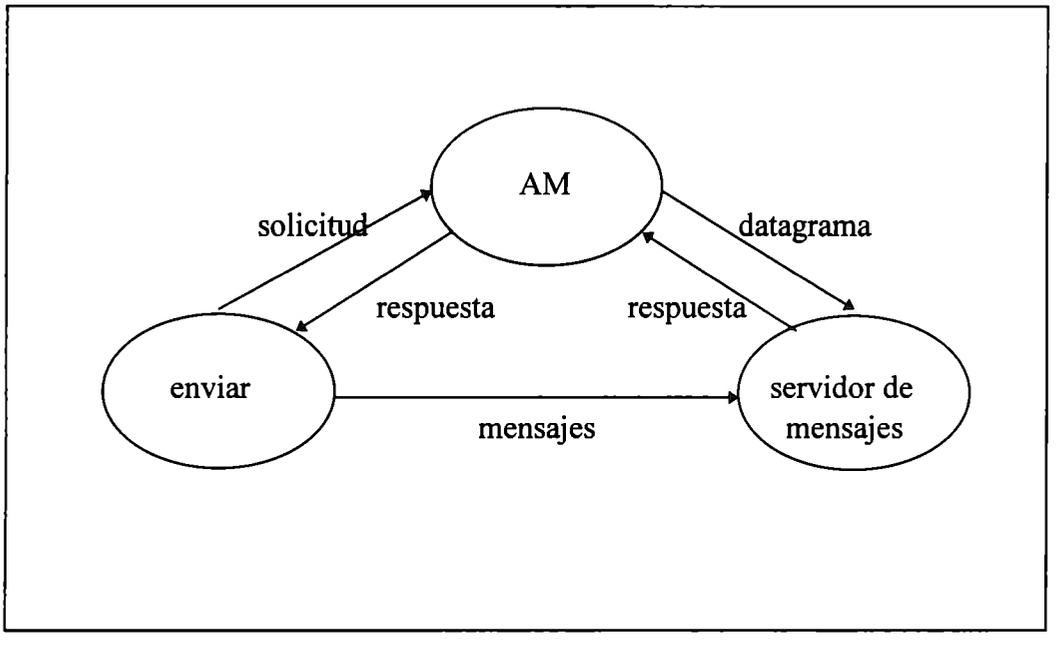

Fig. 7.5.3 Flujo de datos en el sistema.

El código de recepción de mensajes (en el servidor de mensajes o proceso *servidor* simplemente) está sincronizado con el de transmisión de mensajes en el cliente (proceso *enviar*). La finalización de la transmisión ocurre cuando el usuario teclea sólo un INTRO. Esto es representado en el código como una transmisión de un único carácter.
La interacción entre los programas del sistema se puede ver en la figura 7.5.3. Es importante aclarar que la comunicación entre dos procesos se realiza utilizando únicamente un canal.

Cuando ya fue autorizada la comunicación, la conexión se realiza desde el cliente al servidor utilizando un puerto distinto al que se utilizó al hacer la solicitud.
Cada máquina en la red tiene asociado un puerto para recibir información. La tabla de puertos por host es mostrada en la tabla 7.3.1.
La tabla de puertos es implementada como una tabla externa al programa *enviar*. El archivo donde se encuentra esta tabla se denomina *puertos.adm* y es accesado por el programa *am* (Autoridad Multidominio).
La evaluación final es la que realiza la política local en el host destino. Esta evaluación utiliza los datos recibidos en el datagrama provenientes de la AM y lleva a cabo la evaluación local.
La AM utiliza la tabla de puertos para determinar la transferencia de información. Para enviar datos al servidor de mensajes utiliza un puerto específico y por ese mismo puerto recibe la respuesta. La AM debe utilizar el mismo puerto para recibir información del cliente y para enviarle la respuesta. Este mensaje es la respuesta de aprobación o negación de la comunicación. El cliente recibe el mensaje como una variable *cadena*. El envío de la respuesta se hace de la siguiente manera:

```
byteswritten = u_write(communfd,buf,1024);
```

donde la variable *byteswritten* contiene los bytes que se envían al cliente y *u_write* es una función que permite realizar la transferencia de datos. En este caso ya no se usa el datagrama principal de la AM, ya que únicamente se requiere el envío de un mensaje, el cual podrá ser de aprobación o negación dependiendo de la evaluación de la Multipolítica.

```
strcpy(buf,respuesta);

printf("\n Enviando respuesta al cliente...");
byteswritten = u_write(communfd,buf, 1024);
if (byteswritten < 0)
  fprintf(fpant,"Error en el envio de respuesta \n");
```

Fig. 7.5.4 Envío de respuesta de AM al cliente

En lo que se refiere al código del hijo, éste se encuentra dentro de un ciclo while. Este ciclo tiene la función de "escuchar" en un puerto determinado si llega alguna petición. El código del hijo contiene la llamada a la función *atiende*.
Esta función, como se mencionó anteriormente, es en la que se realiza la recepción del datagrama proveniente del cliente y la transferencia de éste al servidor de mensajes. Además en esta función se realiza la transferencia de la respuesta que contiene el mensaje para indicar la aprobación o negación de la transferencia.

```
if (child==0)
 {
  u_close(listenfd);
  atiende(communfd);
  u_close(communfd);
  exit(0);
 }
```

Fig. 7.5.5 Código del hijo de AM

El parámetro que aparece en la función *atiende* es un descriptor de archivo donde se mandará la información. El código fundamental del hijo es presentado en la figura 7.5.5 y puede incluir sentencias adicionales, por ejemplo, impresiones de mensajes para observar cuál es el proceso que atiende la conexión y quién es el cliente.

De esta manera la AM crea un proceso hijo por cada solicitud que recibe. El ciclo *while* pertenece al código del padre, por lo cual, la AM después de crear un hijo para atender a un cliente permanece en dicho ciclo para aceptar más solicitudes de diversos clientes.

La comunicación del proceso cliente con el servidor de mensajes se realiza haciendo una petición de servicio desde el mismo cliente. La conexión entre el cliente y el servidor de mensajes ocurre después de que la AM y la política local han autorizado la comunicación. El servidor de mensajes es un proceso que corre en segundo plano en cada host de la red.

7.6 POLÍTICAS LOCALES

Después de que la AM ha realizado su trabajo, le corresponde a la política local autorizar o rechazar la solicitud de transferencia de mensajes. La evaluación en cada dominio se lleva a cabo en primer lugar con la política MLS, después con la Comercial y por último con la Financiera.

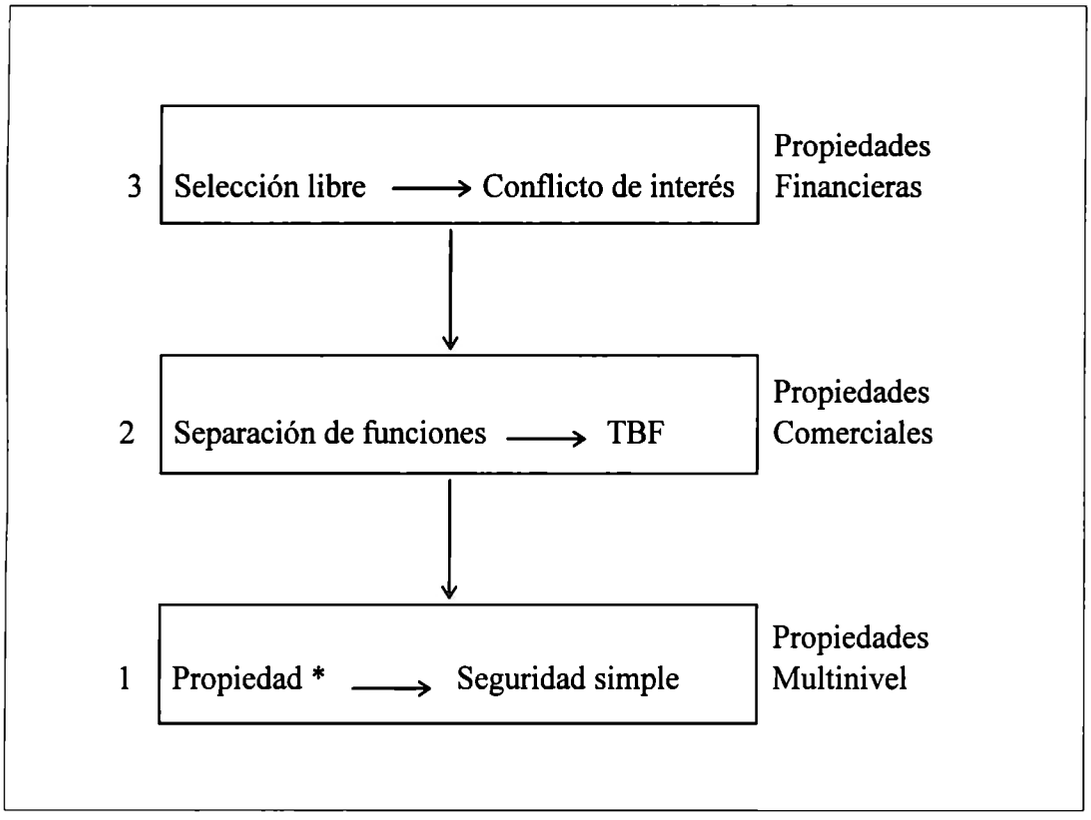

Fig. 7.6.1 Orden de evaluación de políticas

Para poder evaluar la política Financiera se requiere haber evaluado la Comercial. Para evaluar esta última, se necesita haber evaluado la Multinivel (F(C(MLS))).

El programa que implementa la política local en cada host contiene una función para cada una de las políticas individuales MLS, C y F. La implementación de las políticas locales no es abordada en este documento pero puede verse en detalle en el trabajo de tesis de Angélica Ramírez [3].
Dentro del presente trabajo, **las políticas locales son evaluadas en el servidor de mensajes**. La decisión para realizar dos funciones en este programa fue que ambas están directamente relacionadas. La primera función, que es la evaluación, es el punto final en cuanto a la autorización de la comunicación (política local). La segunda acción es la recepción de mensajes. Si esta interacción es aprobada, el servidor de mensajes ejecutará el código para recepción de mensajes.

Los datos que obtiene el programa *servidor* en la recepción del datagrama son indispensables para poder realizar la evaluación.
La forma en que el servidor de mensajes recibe el datagrama proveniente de la AM se muestra en la siguiente línea.

```
bytesread=u_read(communfd, &data, sizeof(struct datag));
```

La búsqueda de la clasificación de un usuario se realiza con el código de la figura 7.6.2 y es similar a las búsquedas de los programas de administración del sistema. En el código de la figura 7.6.2 se realiza una lectura de los usuarios en archivo *usa*, hasta que se encuentra el usuario correcto o bien, se llega al final del archivo.

```
while( strcmp(data.us_origen,us_d_clas_mls) && ! feof(usa) )
 {
  posicion=ftell(usa);
  fseek(usa,posicion+39,0);
  fread(us_d_clas_mls, sizeof(us_d_clas_mls), 1, usa);
 }
```

Fig. 7.6.2 Código de búsqueda de usuario

En caso de que se encuentren los usuarios indicados, es decir, los usuarios que vienen en el datagrama como origen y destino, se toma la clasificación para MLS asociada a cada uno de ellos. Para realizar la evaluación en MLS, la política local asigna un valor numérico a cada atributo. Los atributos que se manejan en este punto son tanto de la información como del usuario. Esto se puede ver en la figura 7.6.3.

Los valores de los atributos son "mapeados" a números enteros con los que se lleva a cabo la comparación de estos atributos (atributos de la información y clasificación de usuarios).
La determinación de la comunicación en la política MLS se realiza comparando los valores numéricos correspondientes a los atributos contra los valores numéricos de la clasificación de la información.

Para la evaluación en la política Comercial se debe verificar en primer lugar si se trata de datos restringidos. Si es así se revisa que el usuario destino esté autorizado para aceptar CDI's. Los Conjuntos de Datos Restringidos son manipulados por Procedimientos de transformación bien definidos. El sistema verifica que tanto el usuario origen como destino posean el TP adecuado para el envío y recepción de mensajes respectivamente.

```
us_d_clas=data.us_d_clas_mls;
switch(*us_d_clas)
 {
  case 'u': us_cl=0;
            break;
  case 'c': us_cl=1;
            break;
  case 's': us_cl=2;
            break;
  case 't': us_cl=3;
 }

info_clas=data.mls;
switch(*info_clas)
 {
  case 'u': info_cl=0;
            break;
  case 'c': info_cl=1;
            break;
  case 's': info_cl=2;
            break;
  case 't': info_cl=3;
 }
```

Fig. 7.6.3 Valores de atributos de información y usuarios

Aún en el caso de que la información que llega es no restringida (UDI), la política local permitirá la interacción en este nivel siempre que los TP's correctos estén asociados a los usuarios que intervienen en la comunicación.

Los datos necesarios para la política Comercial son el *objeto=mensaje* y los procedimientos de transformación que permitan el envío y recepción de mensajes. Estos TP's son *enviar* y *servidor* respectivamente.

Para la evaluación realizada por F se necesita el conjunto de datos de compañía (CDC) y la clase de conflicto de interés (CCI). El sistema compara el Conjunto de Datos de Compañía y la Clase de Conflicto de Interés de ambos usuarios. Esta información es leída previamente por el cliente (programa *enviar*) y transferida a la AM, que a su vez la transfiere a al servidor de mensajes.
Una parte de la función del servidor de mensajes que implementa esta política aparece en la figura 7.6.4

```
if (strcmp(data.us_d_cci,data.cci) || !strcmp(data.us_d_cdc,data.cdc))
        printf("\n Mensaje aceptado fin");
 else
        printf("\n Mensaje no aceptado ");
```

Fig. 7.6.4 Decisión para F

Esto se ejecuta después de consultar el CDC y CCI del origen y destino en el datagrama recibido.

```
if (evalua==1)
 strcpy(respuesta,"OK Transmisión aprobada!");

if (evalua==0)
 strcpy(respuesta, "Transmision negada!");

 strcpy(buf,respuesta);
 buf[80] = '\0';
 byteswritten = u_write(communfd,buf, 1024);
 if (byteswritten < 0)
  printf("Error en el envio de respuesta \n");
 else
  printf("\n Respuesta enviada ");
```

Fig. 7.6.5 Envío de respuesta a la AM

Después de la evaluación del servidor local, éste realizara el envío de la respuesta a la AM. La respuesta será un mensaje "Transmisión aprobada" o "Transmisión negada" dependiendo del resultado de la aplicación de la política. La parte del servidor local que realiza esta función aparece en la figura 7.6.7.

7.7 TRANSFERENCIA DE MENSAJES

Para realizar la transferencia de mensajes se usa una conexión directa entre el cliente y el servidor de mensajes. Hay que distinguir este proceso del que realiza la AM. La Multipolítica recibe una petición del cliente y hace la evaluación para enviar la respuesta. Cuando ya ha sido aprobada la comunicación, el cliente ejecuta el código para envío de mensajes. Este código está sincronizado con el de recepción de mensajes del servidor en la máquina destino.

El cliente realiza la llamada a un segundo cliente sin crear un proceso hijo, ya que no se necesitará la recepción de mensajes hasta que se presente una nueva solicitud Cuando termina la transferencia, el segundo cliente llega a su fin. El servidor de mensajes permanece en ejecución bloqueado hasta que llega otra solicitud. El envío de mensajes se realiza con código de la figura 7.7.1, el cual pertenece a un programa llamado *cliente.c.*

En el código de la figura 7.7.1 hay que destacar que el ciclo es infinito, es decir, no hay una variable cuyo valor controle la salida. La salida ocurre cuando sólo se detecta un ENTER. De esa manera el segundo proceso *cliente* finaliza. En el lado del servidor, existe una línea del programa que revisa si se recibieron datos o solo un ENTER. Si no se reciben datos, el *servidor de mensajes* (el receptor de mensajes) termina de atender al cliente.

```
for( ; ;)
 {
  bytesread = read(STDIN_FILENO, buf, BLKSIZE);
  if ((bytesread == -1) && (errno = EINTR) )
    fprintf(stderr,"Cliente restableciendo lectura \n");
  else
    if (bytesread <= 0)
     break;
    else
    {
     byteswritten = u_write(outfd, buf, bytesread);
      printf("\n bytes escritos %d",byteswritten);
     if (byteswritten != bytesread)
      {
       fprintf(stderr, "Error escribiendo %d bytes, %ld bytes
       escritos\n",
       (long)bytesread, (long)byteswritten);
       break;
      }
   }}
```

Figura 7.7.1 Envío de mensajes

El servidor de mensajes contiene las instrucciones que son contraparte de las del cliente que envía los mensajes, es decir, el servidor de mensajes (programa *servidor*) contiene

```
portnumber = (unsigned short) atoi(argv[1]);
 if ((listenfd = u_open(portnumber)) < 0)
 {   u_error("No se pudo establecer un puerto de conexion\n");
     exit(1);  }
for(;;)
{  if ((communfd = u_listen(listenfd, remote)) < 0)
 {   u_error("Falla en el intento de escuchar \n");
     exit(1);  }
  fprintf(stderr,"La conexion ha sido realizada a %s\n", argv[1]);
  bytesread=u_read(communfd, &data, sizeof(struct datag));
  evalua_mls=pol_mls();
  evalua_com=pol_com();
  evalua_fin=pol_fin();
 /*  Envia respuesta a la AM  */
if (evalua_mls==1 &&  evalua_com==1 && evalua_fin ==1 )
 strcpy(respuesta,"OK Transmision aprobada!");
else
 strcpy(respuesta,"Transmision negada!");
strcpy(buf,respuesta);
buf[80] = '\0';
byteswritten = u_write(communfd,buf, 1024);
if (byteswritten < 0)
printf("Error en el envio de respuesta \n");
printf("\n Respuesta enviada \n ");
if ( !strcmp(respuesta,"OK Transmision aprobada!"))
{  if ((communfd = u_listen(listenfd, remote)) < 0)
   {     u_error("Falla en el intento de escuchar \n");
         exit(1);   }
  fprintf(stderr,"La conexion ha sido realizada a %s\n", argv[1]);
  while( (bytesread = u_read(communfd, buf, BLKSIZE)) > 1)
   {
    byteswritten = write(STDOUT_FILENO, buf, bytesread);
    if (bytesread != byteswritten)
     {
       fprintf(stderr, "Error escribiendo %d bytes, %ld bytes escritos\n",
           (long)bytesread, (long)byteswritten);
       break;
     }
    }
}
```

Fig. 7.7.2 Parte del código del servidor de mensajes.

las instrucciones para la recepción de mensajes y el cliente (programa *cliente*) las instrucciones para el envío de datos. El programa *enviar*, que es el que realiza la solicitud de envío de mensajes, ejecuta una llamada al sistema (*execv*) para correr el programa *cliente*. Éste último es el que efectúa la transferencia de mensajes hacia el servidor que debe estar en ejecución en la máquina destino. Parte del código del servidor de mensajes se puede ver en la figura 7.7.2.

En la figura 7.7.3 se muestra el código del programa *cliente*.

```
portnumber = (unsigned short) atoi(argv[2]);
 if ((outfd = u_connect(portnumber, argv[1])) < 0) {
  u_error("Incapaz de establecer una conexion Internet");
  exit(1);
 }
fprintf(stderr,"La conexion ha sido realizada a %s\n", argv[1]);
strcpy(buf,"");
for( ; ;)
 {
  bytesread = read(STDIN_FILENO, buf, BLKSIZE);
  if  ((bytesread == -1) && (errno = EINTR) )
    fprintf(stderr,"Cliente reestableciendo lectura \n");
  else
    if (bytesread <= 0)
     break;
    else
      if (bytesread==1)
        {
         printf("\n Fin...\n");
         break;
        }
    else {
          byteswritten = u_write(outfd, buf, bytesread);
              if (byteswritten != bytesread)
           {
            fprintf(stderr, "Error escribiendo %d bytes, %ld bytes escritos\n",
            (long)bytesread, (long)byteswritten);
             break;
           }
 }
```

Fig. 7.7.3 Código del cliente para envío de mensajes

7.8 JERARQUÍA DE PROPIEDADES EN LA IMPLEMENTACIÓN

La jerarquía de propiedades para la implementación es la misma que la presentada en capítulos anteriores. Esto se debe a que se están utilizando las tres políticas MLS, C y F que son las que se toman para ejemplificar dicha jerarquía.
Una observación importante es que se maneja únicamente un *objeto* y dos TP's para C. Estos dos TP's son *enviar* y *servidor*. Recordemos que el primero es el programa que permite hacer la solicitud de envío de mensajes y el segundo es el que permite la recepción de datos. Para que un usuario tenga derecho de enviar y recibir mensajes, es necesario que tenga asociados los TP's *enviar* y *servidor*. Existe la posibilidad de que un usuario quiera transmitir un mensaje a otro usuario que no tenga derecho de recibirlo. Este derecho es el que se otorga en la política C con la asignación del TP correspondiente. En otras palabras, se puede ver que para que la comunicación se lleve a cabo es necesario que el origen tenga derecho de ejecución sobre el programa *enviar* y además que el usuario destino tenga derecho también, de ejecución, sobre el programa *servidor*.

Jerarquía utilizada en la implementación:

- Control de flujo de información no saneada
- Propiedad de opción libre
- Propiedad de conflicto de intereses
- Propiedad de separación de deberes
- Propiedad de transacción bien-formada
- Propiedad *
- Propiedad de seguridad simple

7.9 GENERACIÓN DE INFORMACIÓN DE AUDITORIAS

El sistema elabora un reporte de las actividades realizadas para la autorización de la interacción desde el punto de vista de la Multipolítica y de la política local. Este reporte se construye con mensajes grabados en un archivo llamado *reporte.adm*. Estos mensajes son resultados de los diferentes pasos en la evaluación. Los datos del reporte de auditoría son guardados en *reporte.adm* por la AM. Un fragmento de código que realiza reporte es mostrado en la figura 7.9.1.
En el código del programa mostrado en la figura 7.9.1 se observa la apertura del archivo *reporte.adm,* en el cual se guardan los datos obtenidos durante el proceso de evaluación. Después se muestra la escritura de un encabezado y de los datos del host del usuario origen. La escritura de los demás datos de usuarios, tanto del origen como del destino, se han omitido en esta sección debido a que es muy larga y es similar a la presentada en la figura 7.9.1.

```
if ((reporte = fopen("reporte.adm", "a+"))== NULL)
  {
    fprintf(stderr, "Imposible abrir archivo.\n");
    return 1;
  }

/*  Datos del origen que se guardaran en el reporte de auditoria  */

  for(i=0;i<=79;i++)
    mensaje[i]=' ';
  for(i=0;i<=79;i++)
    mensaje[i]='-';
  rango=rango+80;
  fseek(reporte,rango,0);
  fwrite(mensaje, sizeof(mensaje), 1, reporte);
  for(i=0;i<=79;i++)
    mensaje[i]=' ';
  strcpy(mensaje,"    ATRIBUTOS DEL USUARIO ORIGEN");
  rango=rango+80;
  fseek(reporte,rango,0);
  fwrite(mensaje, sizeof(mensaje), 1, reporte);

  for(i=0;i<=79;i++)
    mensaje[i]=' ';
  for(i=0;i<=79;i++)
    mensaje[i]='-';
  rango=rango+80;
  fseek(reporte,rango,0);
  fwrite(mensaje, sizeof(mensaje), 1, reporte);
  strcpy(mensaje,"Host origen: ");
  strcat(mensaje,data.host_origen);
  largo=strlen(mensaje);
  for(i=largo+1;i<=79;i++)
    mensaje[i]=' ';
  rango=rango+80;
  fseek(reporte, rango, 0);
  fwrite(mensaje, sizeof(mensaje), 1, reporte);
```

Fig. 7.9.1 Creación del reporte de auditoría

El programa completo de la AM, que contiene la elaboración de auditorías, se incluye en el anexo 2 .

En la figura 7.9.2 se muestra un ejemplo del reporte obtenido por el sistema, después de una sesión donde se realizaron una serie de interacciones entre dominios de seguridad diferentes.

```
**************************************************
            REPORTE DE AUDITORIA
**************************************************
OK Transmision aprobada!
--------------------------------------------------------------------------------
ATRIBUTOS DEL USUARIO ORIGEN
--------------------------------------------------------------------------------
Host origen: barbados
Usuario origen: al310477
Politica Multinivel presente
Categoria Multinivel para el origen: t
Clasificacion Multinivel para el origen: t
Politica Comercial no presente
Objeto :0
tp 1: 0
tp 2: 0
Politica Financiera presente
CCI : educacion
CDC :unam
--------------------------------------------------------------------------------
ATRIBUTOS DEL USUARIO DESTINO
--------------------------------------------------------------------------------
Host destino: barbados
Usuario destino: al310478
Politica Multinivel presente
Categoria Multinivel para usuario destino: u
Clasificacion Multinivel para usuario destino: u
Politica Comercial no presente
Objeto: 0
tp 1: 0
tp 2: 0
Politica Financiera presente
CCI: educacion
CDC: unam
--------------------------------------------------------------------------------
ATRIBUTOS DE LA INFORMACION
--------------------------------------------------------------------------------
Atributo en MLS : u
Atributo en C : 0
Atributo en F : s
**************************************************
```

Fig. 7.9.2 Reporte de auditoría

En la figura anterior se incluyen los atributos de los usuarios origen y destino, así como de la información. En el este ejemplo la transmisión ha sido aprobada, lo cual se puede ver en mensaje que aparece después del encabezado del reporte. Los atributos que contienen un 0 indican que esa política no está contenida en el dominio del usuario o que no existe esa política en el dominio del origen, esto último, en el caso de atributos de información.

En el ejemplo de la figura anterior, el usuario al310477 solicitó establecer comunicación con al310478. Ambos usuarios pertenecen al mismo dominio llamado *barbados* y contienen el mismo conjunto de políticas, las cuales son; MLS y F. La política C no esta presente en este dominio. Los atributos de la información indican que se trata de datos no confidenciales para ambas políticas, ***u*** para MLS y ***s*** para F. También es importante destacar que ambos usuarios pertenecen a la misma CCI y al mismo CDC ya que están contenidos en el mismo dominio, por lo cual la comunicación siempre se permitirá en la política F. No sucede lo mismo para MLS, ya que en caso de que el usuario origen ponga una etiqueta ***t*** a la información, la comunicación no será permitida porque el usuario destino tiene una clasificación menor (u), lo que quiere decir que al310478 sólo puede recibir información no confidencial en MLS.

7.10 TRABAJOS FUTUROS DE INVESTIGACION

El tema de Seguridad Multidominio no ha sido estudiado exhaustivamente hasta el día de hoy. Se puede profundizar en las investigaciones sobre un sistema de Seguridad Mutidominio Distribuido. En un sistema como este la Autoridad Multidominio procesará los datos de los usuarios de manera distribuida, a diferencia de la AM centralizada del presente trabajo. Se requiere también implantar el sistema Multidominio en un sistema operativo seguro donde existan canales confiables, la información sea transmitida de manera encriptada, exista un fuerte control de acceso, etc. Una posibilidad para desarrollar el trabajo expuesto en esta tesis en un sistema distribuido puede ser a través de Internet. En este caso se tienen que definir las políticas que respetará cada usuario incorporado en el sistema, tal y como se hizo con MLS, C y F. Se tienen que definir también los lugares geográficos donde correrá la AM y quizá desarrollar el servidor de mensajes nuevamente para correrlo a través de una llamada remota.

8 CONCLUSIONES

En el presente trabajo se ha desarrollado un sistema para transmitir información entre dominios de seguridad heterogéneos. Se han realizado pruebas exhaustivas utilizando el enfoque más restrictivo conocido. Este enfoque es propuesto en [1]. Se demostró experimentalmente que el no permitir la traducción de atributos para información no confidencial en la interacción entre dominios distintos es factible. Las condiciones que se deben de cumplir para establecer la comunicación son muy restrictivas pero este hecho asegura que la información no será divulgada a individuos que no tengan autorización para conocerla. El hecho de haber implementado este enfoque utilizando las políticas Multinivel, Comercial y Financiera nos garantiza que las interacciones pueden realizarse con éxito tomando en consideración la confidencialidad y la integridad de la información, incluso, una combinación entre éstas como se expresa en la política Financiera de Brewer y Nash.

Para la implementación del sistema de interacción se han asumido algunas consideraciones sobre el sistema en el que debe actuar. En estas consideraciones se incluye el hecho de que el sistema operativo es seguro, que la información no puede ser accesada durante la transferencia, que no existen canales ocultos y en general todo lo que pueda interferir con la seguridad de la transferencia de la información. El trabajo desarrollado sólo se enfoca a la interacción segura entre dominios con políticas de seguridad distintas.

Actualmente se puede observar en todo el mundo una preocupación sobre temas de seguridad computacional, se desarrollan trabajos teóricos y prácticos, se organizan conferencias internacionales, se divulgan los avances en este campo en libros y revistas especializadas, etc.. En México aún no se tiene una cultura amplia sobre seguridad en cómputo. Podemos observar empresas, instituciones educativas, gubernamentales, etc. que cuentan con una seguridad mínima en sus sistemas y en ocasiones esta seguridad es nula. Estos sistemas son susceptibles de ser violados por personas ajenas a ellos. Las instituciones financieras son las que cuentan con una mayor seguridad en sus sistemas y ello se debe a que su información es sumamente delicada. Una violación en los sistemas financieros puede resultar en pérdidas cuantiosas. Pero aunque estas últimas instituciones mencionadas cuentan con sistemas de seguridad, hay que destacar que muchos de éstos no son desarrollados en las propias instituciones, incluso dentro de nuestro propio país, por lo que la seguridad está en duda.

Por otra parte, para que las interacciones entre dominios de seguridad distintos se lleven a cabo en el mundo real, deben presentarse primero propuestas adecuadas para mantener la seguridad de la información. Los avances en este tema deben aparecer en primer lugar en trabajos teóricos seguidos del desarrollo de sistemas que utilicen tales conceptos. Cuando se garantice una comunicación segura no existirán restricciones para que se establezca una interacción importante de manera global. Actualmente la mayoría de la información que se accesa en sistemas abiertos es información no confidencial.

El desarrollo de la implantación de un sistema de seguridad interdominio es un tema que permite profundizar más. La investigación en relación a la seguridad en la comunicación donde intervengan diversos dominios aún tiene mucho camino por recorrer. El mundo exige cada vez más una comunicación global por lo que es de suponer que dentro de algún tiempo este tema será desarrollado a tal grado que existirá una interacción amplia.

Las interacciones entre dominios de seguridad diferentes utilizando el enfoque descrito en [1] son factibles de llevarse a la práctica. Para ello es necesario que los dominios que interactuen definan políticas similares. Esto se debe a que políticas distintas no podrán ser evaluadas. Un aspecto interesante de esto sería el establecimiento de una política para interacciones seguras en INTERNET, en la que cualquier usuario que desee establecer comunicación con otros dominios tenga forzosamente que adoptar la misma política.

De lo anterior podemos decir que para establecer una comunicación global, es necesario que existan políticas de seguridad estándar, ya que en realidad existen tantas políticas como instituciones.

REFERENCIAS

[1] J. Vázquez-Gómez, Modelling Multidomain Security SIGSAC, New Security Paradigm workshop IEEE Computer Society Press 1993.

[2] J. Vázquez-Gómez, Multidomain Security, Computers & Security, 13 (1994) 161-184.

[3] Salvador Contreras - Angélica Ramírez, Políticas de Seguridad Computacional, Día Internacional de la Seguridad en Cómputo, IMAS- UNAM, Nov. 1996.

[4] D.E. Bell and L.J. La Padula, Security Computer Systems: Mathematical Foundations and Model, Tech. Rep., MITRE Corp., Bedford MA, 1974.

[5] J.A. Goguen and J. Mesenguer, Security policies and security models, IEEE Symp. Security and Privacy, 1989.

[6] D.D. Clark and D.R. Wilson, A comparison of commercial and military computer security policies, IEEE Symp. Security and Privacy, 1987.

[7] D. Brewer and M. Nash, The Chinese Wall Securiy policy, IEEE Symp. Security and Privacy 1989.

[8] ECMA Security in Open Systems : A Security Framwork, ECMA TR/46, 1988.

[9] ECMA, Security in Open Systems : Data Elements and Service Definitions Satndard ECMA-138, Dec. 1989.

[10] US Dept. of Defense, Department of Defense Trusted Computer System Evaluation Criteria, Rep. DOD 5200.28-STD, 1985.

[11] National Computer Security Center, Trusted Network Interpretation of the TCSEC, NCSC-TG-005, Vers. 1, July 1987.

[12] Service Central de la Securité des Systèmes d'information, Critères d'evaluation de la sécurité des systèmes informatiques (ITSEC), 1990.

[13] ISO, Glossary of Information Technology Security Definitions, International Standards Organization Publication ISO/IECJTCI/SC27 N270, 1991.

[14] H.H. Hosmer, Integrating security policies, Proc. 3rd, RADC Workshop of Multilevel Database Security Castile NY, June 1980.

[15] H.H. Hosmer, Metapolicies I, ACM SIGSAC Rev. (1992).

[16] H.H. Hosmer, The multipolicy model: a working paper, Proc. 4th RADC Multilevel Database Security Workshop, April 1991.

[17] L.J. La Padula, Formal modelling in a generalised framework for access control, Proc. Computer Security Fundation Workshop III, IEEE, June 1990.

[18] H.H. Hosmer, The multipolicy machine, a new paradigm for multilevel secure systems, Proc. NISTIR 4614, June 1991.

[19] H.H. Hosmer and M.D. Abrams, Multipolicy Machine: a new security paradigm, IEEE Symp. Security and Privacy, Poster Session Abstracts, 1992.

[20] J. Glassgow, G. MacEwen and P. Panangaden, A logic for reasoning about security, ACM Trans. Comput. Syst., 10 (3) (Aug. 1992).

[21] J. Glassgow and G. Mac Ewen, Obligation as the basis of integrity especification, Proc. Computer Security Fundations Workshop, Franconia, NH, June 1989.

[22] D. McCullough, A Hwkup theorem for multilevel security, IEEE Trans. Softw. Eng., 16(6) (June 1990).

[23] J. McLean, The algebra of security, Proc. 1988 IEEE Symp. Security and Privacy. Order N850, CS Press, Los Alamitos, CA., 1988, pp. 2-7.

[24] J. Alves-Foss and K. Levit, The verification of secure distributed systems, IEEE COMPCON, 1991.

ANEXO A. REPORTE DE AUDITORÍA

****

REPORTE DE AUDITORIA *

****

OK Transmision aprobada!

ATRIBUTOS DEL USUARIO ORIGEN

Host origen: barbados
Usuario origen: al310477
Politica Multinivel presente
Categoria Multinivel para el origen: t
Clasificacion Multinivel para el origen: t
Politica Comercial no presente
Objeto :0
tp 1: 0
tp2:0
Politica Financiera presente
CCI : educación
CDC : unam

ATRIBUTOS DEL USUARIO DESTINO

Host destino : barbados
Usuario destino: al310478
Politica Multinivel presente
Categoria Multinivel para usuario destino: u
Clasificacion Multinivel para usuario destino: u
Politica Comercial no presente
Objeto: 0
tp 1: 0
tp 2: 0
Politica Financiera presente
CCI: educacion
CDC: unam

ATRIBUTOS DE LA INFORMACION

Atributo en MLS : u
Atributo en C : 0
Atributo en F : s

```
**********************************************************************************
REPORTE  DE  AUDITORIA                                                           *
**********************************************************************************
OK Transmision aprobada!
---------------------------------------------------------------------------------
ATRIBUTOS DEL USUARIO ORIGEN
---------------------------------------------------------------------------------
Host origen: barbados
Usuario origen: al310478
Politica Multinivel presente
Categoria Multinivel para el origen: u
Clasificacion Multinivel para el origen: u
Politica Comercial no presente
Objeto :0
tp 1: 0
tp 2: 0
Politica Financiera presente
CCI : educacion
CDC :unam
---------------------------------------------------------------------------------
ATRIBUTOS DEL USUARIO DESTINO
---------------------------------------------------------------------------------
Host destino: barbados
Usuario destino: al310477
Politica Multinivel presente
Categoria Multinivel para usuario destino: t
Clasificacion Multinivel para usuario destino: t
Politica Comercial no presente
Objeto: 0
tp 1: 0
tp 2: 0
Politica Financiera presente
CCI: educacion
CDC: unam
---------------------------------------------------------------------------------
ATRIBUTOS DE LA INFORMACION
---------------------------------------------------------------------------------
Atributo en MLS : u
Atributo en C : 0
Atributo en F : n
**********************************************************************************
```

```
*****************************************************************************
                REPORTE DE AUDITORIA                *
*****************************************************************************
Transmision negada!
-----------------------------------------------------------------------------
ATRIBUTOS DEL USUARIO ORIGEN
-----------------------------------------------------------------------------
Host origen: cuba
Usuario origen: al310473
Politica Multinivel presente
Categoria Multinivel para el origen: t
Clasificacion Multinivel para el origen: t
Politica Comercial presente
Objeto :mensaje
tp 1: enviar
tp 2: servidor
Politica Financiera presente
CCI : seguros
CDC :tepeyac
-----------------------------------------------------------------------------
ATRIBUTOS DEL USUARIO DESTINO
-----------------------------------------------------------------------------
Host destino: guyana
Usuario destino: al310466
Politica Multinivel no presente
Categoria Multinivel para usuario destino: 0
Clasificacion Multinivel para usuario destino: 0
Politica Comercial no presente
Objeto: 0
tp 1: 0
tp 2: 0
Politica Financiera presente
CCI: turismo
CDC: med
-----------------------------------------------------------------------------
ATRIBUTOS DE LA INFORMACION
-----------------------------------------------------------------------------
Atributo en MLS : u
Atributo en C : udi
Atributo en F : s
*****************************************************************************
```

```
**********************************************************************************
REPORTE DE AUDITORIA                    *
**********************************************************************************
OK Transmision aprobada!
---------------------------------------------------------------------------
   ATRIBUTOS DEL USUARIO ORIGEN
---------------------------------------------------------------------------
Host origen: canada
Usuario origen: al310476
 Politica Multinivel presente
Categoria Multinivel para el origen: s
Clasificacion Multinivel para el origen: s
Politica Comercial no presente
Objeto :0
tp 1: 0
tp 2: 0
Politica Financiera presente
CCI : seguros
CDC :tepeyac
---------------------------------------------------------------------------
 ATRIBUTOS DEL USUARIO DESTINO
Host destino: argentina
Usuario destino: al310459
Politica Multinivel presente
Categoria Multinivel para usuario destino: t
Clasificacion Multinivel para usuario destino: t
Politica Comercial no presente
Objeto: 0
tp 1: 0
tp 2: 0
Politica Financiera presente
CCI: financiera
CDC: promex
---------------------------------------------------------------------------
ATRIBUTOS DE LA INFORMACION
---------------------------------------------------------------------------
Atributo en MLS : s
Atributo en C : 0
Atributo en F : s
**********************************************************************************
```

****
REPORTE DE AUDITORIA *
****
OK Transmision aprobada!

ATRIBUTOS DEL USUARIO ORIGEN

Host origen: brasil
Usuario origen: al310457
Politica Multinivel presente
Categoria Multinivel para el origen: s
Clasificacion Multinivel para el origen: s
Politica Comercial no presente
Objeto :0
tp 1: 0
tp 2: 0
Politica Financiera no presente
CCI : 0
CDC :0

ATRIBUTOS DEL USUARIO DESTINO

Host destino: argentina
Usuario destino: al310460
Politica Multinivel presente
Categoria Multinivel para usuario destino: s
Clasificacion Multinivel para usuario destino: s
Politica Comercial no presente
Objeto: 0
tp 1: 0
tp 2: 0
Politica Financiera presente
CCI: financiera
CDC: promex

ATRIBUTOS DE LA INFORMACION

Atributo en MLS : c
Atributo en C : 0
Atributo en F : 0
****

```
**********************************************************************************
REPORTE DE AUDITORIA                    *
**********************************************************************************
OK Transmision aprobada!
---------------------------------------------------------------------------------
ATRIBUTOS DEL USUARIO ORIGEN
---------------------------------------------------------------------------------
Host origen: colombia
Usuario origen: al310453
Politica Multinivel presente
Categoria Multinivel para el origen: t
Clasificacion Multinivel para el origen: t
Politica Comercial presente
Objeto :mensaje
tp 1: enviar
tp 2: servidor
Politica Financiera no presente
CCI : comercial
CDC :liverpool
---------------------------------------------------------------------------------
ATRIBUTOS DEL USUARIO DESTINO
---------------------------------------------------------------------------------
Host destino: salvador
Usuario destino: al310449
Politica Multinivel presente
Categoria Multinivel para usuario destino: t
Clasificacion Multinivel para usuario destino: t
Politica Comercial presente
Objeto: mensaje
tp 1: enviar
tp 2: servidor
Politica Financiera presente
CCI: salud
CDC: imss
---------------------------------------------------------------------------------
ATRIBUTOS DE LA INFORMACION
---------------------------------------------------------------------------------
Atributo en MLS : t
Atributo en C : cdi
Atributo en F : 0

**********************************************************************************
```

```
**********************************************************************************
REPORTE DE AUDITORIA                    *
**********************************************************************************
OK Transmision aprobada!
---------------------------------------------------------------------------------
ATRIBUTOS DEL USUARIO ORIGEN
---------------------------------------------------------------------------------
Host origen: colombia
Usuario origen: al310454
Politica Multinivel presente
Categoria Multinivel para el origen: s
Clasificacion Multinivel para el origen: s
Politica Comercial presente
Objeto :mensaje
tp 1: 0
tp 2: servidor
Politica Financiera no presente
CCI : 0
CDC :0
---------------------------------------------------------------------------------
 ATRIBUTOS DEL USUARIO DESTINO
---------------------------------------------------------------------------------
Host destino: salvador
Usuario destino: al310450
Politica Multinivel present
Categoria Multinivel para usuario destino: s
Clasificacion Multinivel para usuario destino: s
Politica Comercial presente
Objeto: mensaje
tp 1: enviar
tp 2: servidor
Politica Financiera presente
CCI: salud
CDC: imss
---------------------------------------------------------------------------------
ATRIBUTOS DE LA INFORMACION
---------------------------------------------------------------------------------
Atributo en MLS : c
Atributo en C : udi
Atributo en F : 0

**********************************************************************************
```

```
**********************************************************************************
                REPORTE DE AUDITORIA                    *
**********************************************************************************
Transmision negada!
---------------------------------------------------------------------------
ATRIBUTOS DEL USUARIO ORIGEN
---------------------------------------------------------------------------
Host origen: uruguay
Usuario origen: al310462
Politica Multinivel presente
Categoria Multinivel para el origen: c
Clasificacion Multinivel para el origen: c
Politica Comercial no presente
Objeto :0
tp 1: 0
tp 2: 0
Politica Financiera presente
CCI : autos
CDC :vw
---------------------------------------------------------------------------
ATRIBUTOS DEL USUARIO DESTINO
---------------------------------------------------------------------------
Host destino: surinam
Usuario destino: al310468
Politica Multinivel no presente
Categoria Multinivel para usuario destino: 0
Clasificacion Multinivel para usuario destino: 0
Politica Comercial presente
Objeto: mensaje
tp 1: 0
tp 2: servidor
Politica Financiera no presente
CCI: 0
CDC: 0
---------------------------------------------------------------------------
ATRIBUTOS DE LA INFORMACION
---------------------------------------------------------------------------
Atributo en MLS : u
Atributo en C : 0
Atributo en F : s
**********************************************************************************
```

****
REPORTE DE AUDITORIA *
****
Transmision aprobada!

ATRIBUTOS DEL USUARIO ORIGEN

Host origen: costarica
Usuario origen: al310481
Politica Multinivel presente
Categoria Multinivel para el origen: c
Clasificacion Multinivel para el origen: c
Politica Comercial presente
Objeto :mensaje
tp 1: enviar
tp 2: servidor
Politica Financiera presente
CCI : financiera
CDC :bancomer

ATRIBUTOS DEL USUARIO DESTINO

Host destino: bolivia
Usuario destino: al310485
Politica Multinivel presente
Categoria Multinivel para usuario destino: t
Clasificacion Multinivel para usuario destino: t
Politica Comercial presente
Objeto: mensaje
tp 1: enviar
tp 2: servidor
Politica Financiera presente
CCI: educacion
CDC: ipn

ATRIBUTOS DE LA INFORMACION

Atributo en MLS : u
Atributo en C : udi
Atributo en F : n
****

```
**********************************************************************************
                REPORTE DE AUDITORIA                    *
**********************************************************************************
Transmision negada!
---------------------------------------------------------------------------
ATRIBUTOS DEL USUARIO ORIGEN
---------------------------------------------------------------------------
Host origen: costarica
Usuario origen: al310481
Politica Multinivel presente
Categoria Multinivel para el origen: c
Clasificacion Multinivel para el origen: c
Politica Comercial  presente
Objeto :mensaje
tp 1: enviar
tp 2: servidor
Politica Financiera presente
CCI : financiera
CDC :bancomer
---------------------------------------------------------------------------
ATRIBUTOS DEL USUARIO DESTINO
---------------------------------------------------------------------------
Host destino: barbados
Usuario destino: al310478
Politica Multinivel  presente
Categoria Multinivel para usuario destino: t
Clasificacion Multinivel para usuario destino: t
Politica Comercial presente
Objeto: mensaje
tp 1: enviar
tp 2: servidor
Politica Financiera  presente
CCI: educacion
CDC: unam
---------------------------------------------------------------------------
ATRIBUTOS DE LA INFORMACION
---------------------------------------------------------------------------
Atributo en MLS : u
Atributo en C : udi
Atributo en F : n
Violacion indirecta
**********************************************************************************
```

ANEXO B
FUNCIONES DE LIBRERÍA USUADAS EN LA COMUNICACIÓN

Para realizar la interacción entre dominios se utilizan funciones contenidas en una librería llamada *uici.h*. Las funciones son las siguientes:

- int u_open(port)
  Esta función regresa un valor entero que es el descriptor de archivos el cual es *bounded* al puerto dado.
  Parámetro puerto. Número de puerto sobre el del "bind"
  Valor devuelto. Regresa el descriptor de archivo si todo sale bien y -1 si hubo algún error.

- int u_listen(fd, hostn)
  Esta función escucha en un puerto específico las peticiones de los clientes.
  Parámetro fd. Descriptor de archivos previamente *bounded* para escuchar el puerto.
  Parámetro hostn. Nombre del host a escuchar.

- int u_connect(port, hostn)
  Esta función inicializa la comunicación con un servidor remoto.
  Parámetro port. Es el número de puerto conocido en el servidor remoto.
  Parámetro hostn. Es una cadena que contiene el nombre de la máquina destino.
  El valor de regreso es un entero. Si todo sale bien, éste valor es el descriptor de archivo usado en la comunicación y -1 si hay algún error.

- int u_close(fd)
  Esta función cierra la comunicación para un descriptor de archivo especificado en *fd.*
  Parámetro fd. Descriptor de archivo de la conexión socket a cerrarse.
  El valor de regreso es el descriptor de archivo y -1 su hubo algún error.

- size_t u_read(fd, buf, size)
  Toma información de un descriptor de archivo que previamente fue abierto por u_open.
  Parámetro fd. Descriptor de archivo.
  Parámetro buf. *Buffer* de salida.
  Parámetro size. Número de bytes a tomar.
  Si no hay error, esta función regresa el número de bytes leídos y -1 si hubo algún error.

- size_t u_write(fd, buf, size)
  Parámetro fd. Descriptor de archivo.
  Parámetro buf. Buffer de salida.
  Parámetro size. Número de bytes a escribir.
  Si todo sale bien esta función devuelve el número de bytes escritos y -1 si hubo algún error.

ANEXO C
PROGRAMAS DEL SISTEMA DE SEGURIDAD MULTIDOMINIO

```
/*

Programa : am.c   Autoridad Multidominio
Autor    : Salvador Contreras H.
Fecha    : Diciembre de 1997
Objetivo : Controla la interacción entre dominios heterogéneos

*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <limits.h>
#include <sys/wait.h>
#include "uici.h"

#define BLKSIZE 1024
#define MAX_CANON 1024

int no_client;
int child;
char respuesta[80];


struct host
        {
          char host[12];
          char mls[2];
          char com[2];
          char fin[2];
        }h_pol,buf;
```

```
struct datag
  {
   int  pol_o_mls;
   int  pol_d_mls;
   int  pol_o_com;
   int  pol_d_com;
   int  pol_o_fin;
   int  pol_d_fin;
   char us_origen[10];
   char us_destino[10];
   char host_origen[12];
   char host_destino[12];
   char mls[10];
   char com[10];
   char fin[10];
   char cci[10];
   char cdc[10];
   char us_o_cat_mls[3];
   char us_d_cat_mls[3];
   char us_o_clas_mls[4];
   char us_d_clas_mls[4];
   char us_o_cci[10];
   char us_d_cci[10];
   char us_o_cdc[10];
   char us_d_cdc[10];
   char us_o_objeto[10];
   char us_d_objeto[10];
   char us_o_tp1[12];
   char us_d_tp1[12];
   char us_o_tp2[12];
   char us_d_tp2[12];
  }data;


int busca_host_mls(char host_name[12])
{
  FILE *hosts;
  int posicion;
  int encontrado;
  int i,rango;
  char mensaje[80];
  FILE *reporte;
```

```
 if ((hosts = fopen("politica.adm", "r+"))== NULL)
 {
   fprintf(stderr, "Imposible abrir archivo.\n");
   return 1;
 }


 fseek(hosts, 0, 0);
 fread(h_pol.host, sizeof(h_pol.host), 1, hosts);
 posicion=ftell(hosts);
 fseek(hosts, posicion, 0);
while( strcmp(host_name,h_pol.host) && ! feof(hosts) )
 {
  posicion=ftell(hosts);
  fseek(hosts,posicion+6,0);
  fread(h_pol.host, sizeof(h_pol.host), 1, hosts);
 }

if ( !strcmp(h_pol.host,host_name) )
 {
  posicion=ftell(hosts);
  fseek(hosts,posicion,0);
  fread(h_pol.mls, sizeof(h_pol.mls), 1, hosts);
  if (strcmp(h_pol.mls,"1"))
   {
    printf("\n No existe mls, transferencia suspendida");
    strcpy(respuesta,"No existe mls, transferencia suspendida");
    encontrado=0;
   }
   else
    {
     encontrado=1;
    }
 }
else
 {
  printf("\n no existe host (mls), tranferencia suspendida");
  strcpy(respuesta,"No existe host, transferencia suspendida");
  encontrado=0;
 }

fclose(hosts);
```

```
 if ((reporte = fopen("reporte.adm", "a+"))== NULL)
 {
   fprintf(stderr, "Imposible abrir archivo.\n");
   return 1;
 }

 for(i=0;i<=79;i++)
   mensaje[i]=' ';
 strcpy(mensaje,respuesta);
 rango=rango+80;
 fseek(reporte,rango,0);
 fwrite(mensaje, sizeof(mensaje), 1, reporte);

 fclose(reporte);

if (encontrado==1)
 return 1;
else
 return 0;
}



/*  Funciones para buscar los host del origen y destino  */

int busca_host_com(char host_name[12])
{
  FILE *hosts;
  int posicion;
  int encontrado;
  int i,rango;
  char mensaje[80];
  FILE *reporte;


 if ((hosts = fopen("politica.adm", "r+"))== NULL)
 {
   fprintf(stderr, "Imposible abrir archivo.\n");
   return 1;
 }

 fseek(hosts, 0, 0);
 fread(h_pol.host, sizeof(h_pol.host), 1, hosts);
 posicion=ftell(hosts);
 fseek(hosts, posicion, 0);
```

```
 while( strcmp(host_name,h_pol.host) && ! feof(hosts) )
  {
   posicion=ftell(hosts);
   fseek(hosts,posicion+6,0);
   fread(h_pol.host, sizeof(h_pol.host), 1, hosts);
  }

 if ( !strcmp(h_pol.host,host_name) )
  {
   posicion=ftell(hosts);
   fseek(hosts,posicion+2,0);
   fread(h_pol.com, sizeof(h_pol.com), 1, hosts);
   encontrado=1;
  }
 else
  {
   printf("\n no existe host, tranferencia suspendida");
   strcpy(respuesta,"No existe host (com), transferencia suspendida");
   encontrado=0;
  }

fclose(hosts);

 if ((reporte = fopen("reporte.adm", "a+"))== NULL)
 {
   fprintf(stderr, "Imposible abrir archivo.\n");
   return 1;
 }

 for(i=0;i<=79;i++)
   mensaje[i]=' ';
 strcpy(mensaje,respuesta);
 rango=rango+80;
 fseek(reporte,rango,0);
 fwrite(mensaje, sizeof(mensaje), 1, reporte);

 fclose(reporte);

if( encontrado==1 &&  !strcmp(h_pol.com,"1") )
   return 1;
else
   return 0;
 }
```

```
int busca_host_fin(char host_name[12])
{
  FILE *hosts;
  int posicion;
  int encontrado;
  int i,rango;
  char mensaje[80];
  FILE *reporte;

 if ((hosts = fopen("politica.adm", "r+"))== NULL)
 {
   fprintf(stderr, "Imposible abrir archivo.\n");
   return 1;
 }
 fseek(hosts, 0, 0);
 fread(h_pol.host, sizeof(h_pol.host), 1, hosts);
 posicion=ftell(hosts);
 fseek(hosts, posicion, 0);
 while( strcmp(host_name,h_pol.host) && ! feof(hosts) )
 {
  posicion=ftell(hosts);
  fseek(hosts,posicion+6,0);
  fread(h_pol.host, sizeof(h_pol.host), 1, hosts);
 }
if ( !strcmp(h_pol.host,host_name) )
 {
  posicion=ftell(hosts);
  fseek(hosts,posicion+4,0);
  fread(h_pol.fin, sizeof(h_pol.fin), 1, hosts);
  encontrado=1;
 }
else
  {
  printf("\n no existe host, tranferencia suspendida");
  strcpy(respuesta,"No existe host (fin) , transferencia suspendida");
  fclose(hosts);
  encontrado=0;
  }
 fclose(hosts);
 if ((reporte = fopen("reporte.adm", "a+"))== NULL)
 {
   fprintf(stderr, "Imposible abrir archivo.\n");
   return 1;
 }
```

```
  for(i=0;i<=79;i++)
    mensaje[i]=' ';
  strcpy(mensaje,respuesta);
  rango=rango+80;
  fseek(reporte,rango,0);
  fwrite(mensaje, sizeof(mensaje), 1, reporte);
  fclose(reporte);
  if (encontrado==1 &&  !strcmp(h_pol.fin,"1") )
    return 1;
  else
    return 0;
 }

 /*  Funciones que verifican las politicas existentes en los nodos
     que interactuan en la comunicacion
 */

int politica_mls()
{
  int ap_origen=0;
  int ap_destino=0;
  int  e_mls = 0;
  ap_origen=busca_host_mls(data.host_origen);
  ap_destino=busca_host_mls(data.host_destino);
  data.pol_o_mls=ap_origen;
  data.pol_d_mls=ap_destino;
 if(ap_origen==1  && ap_destino==1)
  {
   e_mls=1;
  }
  else
  {
    e_mls=0;
  }
 if(e_mls==1)
  return 1;
 else
  return 0;
}
```

```
int politica_comercial()
 {
  int aprobada=0;
  int e_com=0;
  int ap_origen=0;
  int ap_destino=0;

  aprobada=politica_mls();
   if (aprobada==0)
    {
     printf("\n Politica MLS no aprobada \n");
    }
   /* Si la politica mls fue aprobada, continua con la comercial   */
   ap_origen=busca_host_com(data.host_origen);
   ap_destino=busca_host_com(data.host_destino);
   data.pol_o_com=ap_origen;
   data.pol_d_com=ap_destino;
   if (data.pol_o_com==1 && data.pol_d_com==1)
            {
              e_com=1;
            }
   if (data.pol_o_com==1 && data.pol_d_com==0)
          if(!strcmp(data.com,"udi"))
           {
         e_com=1;
           }
          else
           {
              e_com=0;
           }
/*  en caso de que no exista politica com en el origen, la AM debe asignar
    "udi" en el campo de su atributo del datagrama que sera enviado. De
    esta manera el destino se dara cuenta que no existe politica comercial
    en el origen.
*/
   if (ap_origen==0)
    {
     strcpy(data.com,"0");
     e_com=1;
    }
   if (e_com==1 && aprobada==1)
      return 1;
   else
      return 0;
}
```

```
int politica_financiera()
 {
    int aprobada=0;
    int e_fin=0;
    int ap_origen=0;
    int ap_destino=0;

     aprobada=politica_comercial();
     if (aprobada==0)
      {
            printf("\n Politica Comercial no aprobada \n");
      }

  /* Si la politica comercial fue aprobada, continua con la financiera */

  ap_origen=busca_host_fin(data.host_origen);
  ap_destino=busca_host_fin(data.host_destino);

  data.pol_o_fin=ap_origen;
  data.pol_d_fin=ap_destino;

  if (data.pol_o_fin==1 && data.pol_d_fin==1)
   {
    e_fin=1;
   }

  if (data.pol_o_fin==1 && data.pol_d_fin==0)
   if(!strcmp(data.fin,"s"))
    {
     e_fin=1;
    }
   else
    {
     e_fin=0;
    }

 /*  En caso de que no exista politica fin en el origen, la AM debe asignar
    "0" en el campo de su atributo del datagrama que sera enviado. De
    esta manera el destino se dara cuenta que no existe politica fin
    en el origen.
 */

   if(ap_origen==0)
    {
     strcpy(data.fin,"0");
```

```
        e_fin=1;
      }

  if (e_fin==1 && aprobada==1)
     return 1;
  else
     return 0;
 }

/*  Funcion que comienza la verificacion de politicas  */

int busca_politicas()
 {
  int aprobada=0;

  aprobada=politica_financiera();
  if (aprobada==1)
     strcpy(respuesta,"OK Transmision aprobada!");
  else
     strcpy(respuesta,"Transmision negada!");
  if (aprobada != 1)
    return 0;
   else
    return 1;
 }

/*  Funcion que atiende las peticiones de comunicacion  */

int atiende(communfd)
  int communfd;
{
 int bytesread;
 int byteswritten;
 char buf[BLKSIZE];
 FILE *fpant;
 FILE *puertos;
 char resp[80];
 int ap;

 int outfd;
 int alto;
 int portnumber;
 int canal;
 int posicion;
```

```
struct host
        {
          char host[12];
          char puerto[5];
        }pto;

   bytesread=u_read(communfd, &data, sizeof(struct datag));

 /* Busca la politica del origen */

   ap=busca_politicas();
   printf("\n Respuesta : %s",respuesta);
 if (strcmp(respuesta,"OK Transmision aprobada!") )
{
   strcpy(buf,respuesta);

   byteswritten = u_write(communfd,buf, 1024);
   if (byteswritten < 0)
     fprintf(fpant,"Error en el envio de respuesta \n");
   guarda_reporte();
   exit(1);
}
   canal=communfd;

/*
    Envia datagrama al servidor de mensajes,
     lo primero que hace es seleccionar el puerto correcto
*/

   if ((puertos = fopen("puertos.adm", "r+"))== NULL)
   {
     fprintf(stderr, "Imposible abrir archivo.\n");
     return 1;
   }

   fseek(puertos, 0, 0);
   fread(pto.host, sizeof(pto.host), 1, puertos);
   posicion=ftell(puertos);
   fseek(puertos, posicion, 0);
 while( strcmp(data.host_destino,pto.host) && ! feof(puertos) )
   {
    posicion=ftell(puertos);
    fseek(puertos,posicion+12,0);
    fread(pto.host, sizeof(pto.host), 1, puertos);
   }
```

```
 if ( !strcmp(pto.host,data.host_destino ))
  {
   posicion=ftell(puertos);
   fseek(puertos,posicion,0);
   fread(pto.puerto, sizeof(pto.puerto), 1, puertos);
   fclose(puertos);
  }

 printf("\n PUERTO : %s",pto.puerto);

 portnumber = (unsigned short) atoi(pto.puerto);
        if ((outfd = u_connect(portnumber, data.host_destino)) < 0)
         {
          u_error("Falla en puerto, Incapaz de establecer una conexion Internet");
          exit(1);
         }
         byteswritten = u_write(outfd, &data , sizeof(struct datag));
         if (byteswritten < 0)
          {
          fprintf(stderr, "Error escribiendo %s bytes, %ld bytes escritos\n",
          buf, (long)byteswritten);
          }

/*  Recibe respuesta del servidor de mensajes  */

  printf("\n Recibiendo respuesta del servidor de mensajes...");
  while( (bytesread = u_read(outfd, buf, BLKSIZE)) > 1)
  {
   buf[80] = '\0';
   break;
  }
  strcpy(respuesta,buf);

/* Guarda los datos en el reporte de auditoria  */
 guarda_reporte();

/*  Envia respuesta al cliente que solicito la comunicacion (proceso "enviar")

*/

 byteswritten = u_write(communfd,buf, 1024);
 if (byteswritten < 0)
  fprintf(fpant,"Error en el envio de respuesta \n");
}
```

```
int guarda_reporte()
{
 int posicion;
 int rango=1;

 char mensaje[80];
 int i,largo;
 FILE *reporte;


/*  Guarda la respuesta de la evaluaci'on local en el reporte de auditoria */

  if ((reporte = fopen("reporte.adm", "a+"))== NULL)
  {
    fprintf(stderr, "Imposible abrir archivo.\n");
    return 1;
  }

  for(i=0;i<=79;i++)
    mensaje[i]=' ';
  for(i=0;i<=79;i++)
    mensaje[i]='*';
  rango=rango+80;
  fseek(reporte,rango,0);
  fwrite(mensaje, sizeof(mensaje), 1, reporte);

  for(i=0;i<=79;i++)
    mensaje[i]=' ';
  strcpy(mensaje,"*                    REPORTE DE AUDITORIA ");
  rango=rango+80;
  fseek(reporte,rango,0);
  fwrite(mensaje, sizeof(mensaje), 1, reporte);

  mensaje[0]='*';
  for(i=1;i<=79;i++)
    mensaje[i]=' ';
  rango=rango+80;
  fseek(reporte,rango,0);
  fwrite(mensaje, sizeof(mensaje), 1, reporte);

  for(i=0;i<=79;i++)
    mensaje[i]=' ';
  for(i=0;i<=79;i++)
    mensaje[i]='*';
  rango=rango+80;
```

```
 fseek(reporte,rango,0);
 fwrite(mensaje, sizeof(mensaje), 1, reporte);

 for(i=0;i<=79;i++)
   mensaje[i]=' ';
 strcpy(mensaje,respuesta);
 rango=rango+80;
 fseek(reporte,rango,0);
 fwrite(mensaje, sizeof(mensaje), 1, reporte);
 for(i=0;i<=79;i++)
   mensaje[i]=' ';
 for(i=0;i<=79;i++)
   mensaje[i]='*';
 rango=rango+80;
 fseek(reporte,rango,0);
 fwrite(mensaje, sizeof(mensaje), 1, reporte);

 if ((reporte = fopen("reporte.adm", "a+"))== NULL)
 {
   fprintf(stderr, "Imposible abrir archivo.\n");
   return 1;
 }
 for(i=0;i<=79;i++)
   mensaje[i]=' ';
 rango=rango+80;
 fseek(reporte,rango,0);
 fwrite(mensaje, sizeof(mensaje), 1, reporte);

/*  Datos del origen que se guardaran en el reporte de auditoria  */

 for(i=0;i<=79;i++)
   mensaje[i]=' ';
 for(i=0;i<=79;i++)
   mensaje[i]='-';
 rango=rango+80;
 fseek(reporte,rango,0);
 fwrite(mensaje, sizeof(mensaje), 1, reporte);

 for(i=0;i<=79;i++)
   mensaje[i]=' ';
 strcpy(mensaje,"    ATRIBUTOS DEL USUARIO ORIGEN");
 rango=rango+80;
 fseek(reporte,rango,0);
 fwrite(mensaje, sizeof(mensaje), 1, reporte);
```

```
 for(i=0;i<=79;i++)
   mensaje[i]=' ';
 for(i=0;i<=79;i++)
   mensaje[i]='-';
 rango=rango+80;
 fseek(reporte,rango,0);
 fwrite(mensaje, sizeof(mensaje), 1, reporte);
 strcpy(mensaje,"Host origen: ");
 strcat(mensaje,data.host_origen);
 largo=strlen(mensaje);
 for(i=largo+1;i<=79;i++)
   mensaje[i]=' ';
 rango=rango+80;
 fseek(reporte, rango, 0);
 fwrite(mensaje, sizeof(mensaje), 1, reporte);

 for(i=0;i<=79;i++)
   mensaje[i]=' ';
 strcpy(mensaje,"Usuario origen: ");
 strcat(mensaje,data.us_origen);
 rango=rango+80;
 fseek(reporte,rango,0);
 fwrite(mensaje, sizeof(mensaje), 1, reporte);


/*  Datos de la politica Multinivel   */

 for(i=0;i<=79;i++)
   mensaje[i]=' ';
 if(data.pol_o_mls == 1)
   strcpy(mensaje,"Politica Multinivel presente");
 else
   strcpy(mensaje,"Politica Multinivel no presente");
 rango=rango+80;
 fseek(reporte,rango,0);
 fwrite(mensaje, sizeof(mensaje), 1, reporte);


 for(i=0;i<=79;i++)
   mensaje[i]=' ';
 strcpy(mensaje,"Categoria Multinivel para el origen: ");
 strcat(mensaje, data.us_o_cat_mls);
 rango=rango+80;
 fseek(reporte,rango,0);
 fwrite(mensaje, sizeof(mensaje), 1, reporte);
```

```
 for(i=0;i<=79;i++)
   mensaje[i]=' ';
 strcpy(mensaje,"Clasificacion Multinivel para el origen: ");
 strcat(mensaje, data.us_o_clas_mls);
 rango=rango+80;
 fseek(reporte,rango,0);
 fwrite(mensaje, sizeof(mensaje), 1, reporte);


/*  Datos de la politica Comercial  */

 for(i=0;i<=79;i++)
   mensaje[i]=' ';
 if(data.pol_o_com == 1)
   strcpy(mensaje,"Politica Comercial presente");
 else
   strcpy(mensaje,"Politica Comercial no presente");
 rango=rango+80;
 fseek(reporte,rango,0);
 fwrite(mensaje, sizeof(mensaje), 1, reporte);


 for(i=0;i<=79;i++)
   mensaje[i]=' ';
 strcpy(mensaje,"Objeto :");
 strcat(mensaje, data.us_o_objeto);
 rango=rango+80;
 fseek(reporte,rango,0);
 fwrite(mensaje, sizeof(mensaje), 1, reporte);


 for(i=0;i<=79;i++)
   mensaje[i]=' ';
 strcpy(mensaje,"tp 1: ");
 strcat(mensaje, data.us_o_tp1);
 rango=rango+80;
 fseek(reporte,rango,0);
 fwrite(mensaje, sizeof(mensaje), 1, reporte);
 for(i=0;i<=79;i++)
   mensaje[i]=' ';
 strcpy(mensaje,"tp 2: ");
 strcat(mensaje, data.us_o_tp2);
 rango=rango+80;
 fseek(reporte,rango,0);
 fwrite(mensaje, sizeof(mensaje), 1, reporte);
```

```
/* Datos de la politica Financiera   */

  for(i=0;i<=79;i++)
    mensaje[i]=' ';
  if(data.pol_o_fin == 1)
    strcpy(mensaje,"Politica Financiera presente");
  else
    strcpy(mensaje,"Politica Financiera no presente");
  rango=rango+80;
  fseek(reporte,rango,0);
  fwrite(mensaje, sizeof(mensaje), 1, reporte);

  for(i=0;i<=79;i++)
    mensaje[i]=' ';
  strcpy(mensaje,"CCI : ");
  strcat(mensaje, data.us_o_cci);
  rango=rango+80;
  fseek(reporte,rango,0);
  fwrite(mensaje, sizeof(mensaje), 1, reporte);

  for(i=0;i<=79;i++)
    mensaje[i]=' ';
  strcpy(mensaje,"CDC :");
  strcat(mensaje, data.us_o_cdc);
  rango=rango+80;
  fseek(reporte,rango,0);
  fwrite(mensaje, sizeof(mensaje), 1, reporte);


/*  Datos del destino que se guardaran en el reporte de auditoria  */

  for(i=0;i<=79;i++)
    mensaje[i]=' ';
  for(i=0;i<=79;i++)
    mensaje[i]='-';
  rango=rango+80;
  fseek(reporte,rango,0);
  fwrite(mensaje, sizeof(mensaje), 1, reporte);

  for(i=0;i<=79;i++)
    mensaje[i]=' ';
  strcpy(mensaje,"    ATRIBUTOS DEL USUARIO DESTINO");
  rango=rango+80;
  fseek(reporte,rango,0);
  fwrite(mensaje, sizeof(mensaje), 1, reporte);
```

```
  for(i=0;i<=79;i++)
    mensaje[i]=' ';
  for(i=0;i<=79;i++)
    mensaje[i]='-';
  rango=rango+80;
  fseek(reporte,rango,0);
  fwrite(mensaje, sizeof(mensaje), 1, reporte);

  for(i=0;i<=79;i++)
    mensaje[i]=' ';
  strcpy(mensaje,"Host destino: ");
  rango=rango+80;
  strcat(mensaje,data.host_destino);
  largo=strlen(mensaje);
  for(i=largo+1;i<=79;i++)
    mensaje[i]=' ';
  rango=rango+80;
  fseek(reporte, rango, 0);
  fwrite(mensaje, sizeof(mensaje), 1, reporte);

  for(i=0;i<=79;i++)
    mensaje[i]=' ';
  strcpy(mensaje,"Usuario destino: ");
  strcat(mensaje,data.us_destino);
  rango=rango+80;
  fseek(reporte,rango,0);
  fwrite(mensaje, sizeof(mensaje), 1, reporte);


/*  Datos de la politica Multinivel   */

  for(i=0;i<=79;i++)
    mensaje[i]=' ';
  if(data.pol_d_mls == 1)
    strcpy(mensaje,"Politica Multinivel presente");
  else
    strcpy(mensaje,"Politica Multinivel no presente");
  rango=rango+80;
  fseek(reporte,rango,0);
  fwrite(mensaje, sizeof(mensaje), 1, reporte);


  for(i=0;i<=79;i++)
    mensaje[i]=' ';
  strcpy(mensaje,"Categoria Multinivel para usuario destino: ");
```

```
  strcat(mensaje, data.us_d_cat_mls);
  rango=rango+80;
  fseek(reporte,rango,0);
  fwrite(mensaje, sizeof(mensaje), 1, reporte);

  for(i=0;i<=79;i++)
    mensaje[i]=' ';
  strcpy(mensaje,"Clasificacion Multinivel para usuario destino: ");
  strcat(mensaje, data.us_d_clas_mls);
  rango=rango+80;
  fseek(reporte,rango,0);
  fwrite(mensaje, sizeof(mensaje), 1, reporte);


/*  Datos de la politica Comercial  */

  for(i=0;i<=79;i++)
    mensaje[i]=' ';
  if(data.pol_d_com == 1)
    strcpy(mensaje,"Politica Comercial presente");
  else
    strcpy(mensaje,"Politica Comercial no presente");
  rango=rango+80;
  fseek(reporte,rango,0);
  fwrite(mensaje, sizeof(mensaje), 1, reporte);


  for(i=0;i<=79;i++)
    mensaje[i]=' ';
  strcpy(mensaje,"Objeto: ");
  strcat(mensaje, data.us_d_objeto);
  rango=rango+80;
  fseek(reporte,rango,0);
  fwrite(mensaje, sizeof(mensaje), 1, reporte);

  for(i=0;i<=79;i++)
    mensaje[i]=' ';
  strcpy(mensaje,"tp 1: ");
  strcat(mensaje, data.us_d_tp1);
  rango=rango+80;
  fseek(reporte,rango,0);
  fwrite(mensaje, sizeof(mensaje), 1, reporte);

  for(i=0;i<=79;i++)
    mensaje[i]=' ';
```

```
  strcpy(mensaje,"tp 2: ");
  strcat(mensaje, data.us_d_tp2);
  rango=rango+80;
  fseek(reporte,rango,0);
  fwrite(mensaje, sizeof(mensaje), 1, reporte);


/* Datos de la politica Financiera   */

  for(i=0;i<=79;i++)
    mensaje[i]=' ';
  if(data.pol_d_fin == 1)
    strcpy(mensaje,"Politica Financiera presente");
  else
    strcpy(mensaje,"Politica Financiera no presente");
  rango=rango+80;
  fseek(reporte,rango,0);
  fwrite(mensaje, sizeof(mensaje), 1, reporte);


  for(i=0;i<=79;i++)
    mensaje[i]=' ';
  strcpy(mensaje,"CCI: ");
  strncat(mensaje, data.us_d_cci,10);
  rango=rango+80;
  fseek(reporte,rango,0);
  fwrite(mensaje, sizeof(mensaje), 1, reporte);

  for(i=0;i<=79;i++)
    mensaje[i]=' ';
  strcpy(mensaje,"CDC: ");
  strcat(mensaje, data.us_d_cdc);
  rango=rango+80;
  fseek(reporte,rango,0);
  fwrite(mensaje, sizeof(mensaje), 1, reporte);

  for(i=0;i<=79;i++)
    mensaje[i]=' ';
  for(i=0;i<=79;i++)
    mensaje[i]='-';
  rango=rango+80;
  fseek(reporte,rango,0);
  fwrite(mensaje, sizeof(mensaje), 1, reporte);
```

```
  for(i=0;i<=79;i++)
    mensaje[i]=' ';
  strcpy(mensaje,"    ATRIBUTOS DE LA INFORMACION");
  rango=rango+80;
  fseek(reporte,rango,0);
  fwrite(mensaje, sizeof(mensaje), 1, reporte);

  for(i=0;i<=79;i++)
    mensaje[i]=' ';
  for(i=0;i<=79;i++)
    mensaje[i]='-';
  rango=rango+80;
  fseek(reporte,rango,0);
  fwrite(mensaje, sizeof(mensaje), 1, reporte);


  for(i=0;i<=79;i++)
    mensaje[i]=' ';
  strcpy(mensaje,"Atributo en MLS : ");
  strcat(mensaje, data.mls);
  rango=rango+80;
  fseek(reporte,rango,0);
  fwrite(mensaje, sizeof(mensaje), 1, reporte);

  for(i=0;i<=79;i++)
    mensaje[i]=' ';
  strcpy(mensaje,"Atributo en C : ");
  strcat(mensaje, data.com);
  rango=rango+80;
  fseek(reporte,rango,0);
  fwrite(mensaje, sizeof(mensaje), 1, reporte);

  for(i=0;i<=79;i++)
    mensaje[i]=' ';
  strcpy(mensaje,"Atributo en F : ");
  strcat(mensaje, data.fin);
  rango=rango+80;
  fseek(reporte,rango,0);
  fwrite(mensaje, sizeof(mensaje), 1, reporte);
  fclose(reporte);
}
```

```
main(argc, argv)
 int argc;
 char *argv[];
{
  u_port_t portnumber;
  int listenfd;
  int communfd;
  char client[MAX_CANON];
  int bytes_copied;


  no_client = 1;
  if (argc != 2)
  {
   fprintf(stderr, "Uso: %s puerto \n", argv[0]);
   exit(1);
  }
  portnumber = (u_port_t)atoi(argv[1]);
  if ((listenfd = u_open(portnumber)) == -1)
  {
   u_error("No se pudo establecer una conexion de puerto");
   exit(1);
  }
  while ((communfd = u_listen(listenfd, client)) != -1)
  {
   fprintf(stderr, "[%ld]: Una conexion fue establecida con %s \n",
              (long) getpid(), client);
   no_client ++;
   if ((child = fork()) == -1)
   {
     fprintf(stderr, "No se pudo crear un hijo \n");
     break;
   }
   if (child == 0)
   {                        /* Codigo del hijo */
     u_close(listenfd);
     fprintf(stderr, "\n \tProceso [%ld] atendiendo la conexion con %s (%d) \n",
              (long) getpid(), client, no_client);

     atiende(communfd);
     u_close(communfd);
     fprintf(stderr, "\n \tProceso [%ld] ha finalizado de atender a: %s (%d) \n",
              (long) getpid(), client, no_client);
     exit(1);
   }
```

```
    else
    {                       /* Codigo del padre */
      u_close(communfd);
      while (waitpid(-1, NULL, WNOHANG) > 0);
    }
  }
  exit(0);
  return 1;
}  /*   fin   */
```

```
/*

 programa : enviar.c   ( Cliente de AM )
 autor      : Salvador Contreras H.
 fecha      : Diciembre de 1997
 objetivo   : Hacer solicitud de envío de mensajes

*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <errno.h>
#include <sys/types.h>
#include "uici.h"

#define BLKSIZE 1024

 struct datag
   {
    int  pol_o_mls;
    int  pol_d_mls;
    int  pol_o_com;
    int  pol_d_com;
    int  pol_o_fin;
    int  pol_d_fin;
    char us_origen[10];
    char us_destino[10];
    char host_origen[12];
    char host_destino[12];
    char mls[10];
    char com[10];
    char fin[10];
    char cci[10];
    char cdc[10];
    char us_o_cat_mls[3];
    char us_d_cat_mls[3];
    char us_o_clas_mls[4];
    char us_d_clas_mls[4];
    char us_o_cci[10];
    char us_d_cci[10];
    char us_o_cdc[10];
    char us_d_cdc[10];
    char us_o_objeto[10];
```

```
    char us_d_objeto[10];
    char us_o_tp1[12];
    char us_d_tp1[12];
    char us_o_tp2[12];
    char us_d_tp2[12];
    }data;

char n_puerto[80];
char n_host[12];

usr1handler(s)
 int s;
{
 fprintf(stderr,"Senal SIGUR1 recibida \n");
}
usr2handler(s)
 int s;
{
 fprintf(stderr,"Senal SIGUR2 recibida \n");
}
install_usr_handlers()
{
 struct sigaction newact;
 newact.sa_handler = usr1handler;
 sigemptyset(&newact.sa_mask);
 newact.sa_flags = 0;
 if (sigaction(SIGUSR1, &newact, (struct sigaction *) NULL) == -1)
 {
  perror(" No se pudo instalar el handler de la senal ");
  return;
 }
 newact.sa_handler = usr2handler;
 if (sigaction(SIGUSR2, &newact, (struct sigaction *) NULL) == -1)
 {
  perror(" No se pudo instalar el handler de la senal ");
  return;
 }
 fprintf(stderr,"Proceso cliente %ld listo  \n",
    (long)getpid());
}
```

```
main(argc, argv)
 int argc;
 char *argv[];
{
  unsigned short portnumber;
  int outfd;
  size_t bytesread;
  size_t byteswritten;
  char buf[BLKSIZE];
  size_t l_atributos;
  int i,k;
  int communfd;
  char respuesta[80];
  int posicion;
  FILE *puertos;
  FILE *atrib;
  char us_id[10];
  char us_host[12];
  char us_cat_mls[10];
  char us_clas_mls[10];
  char us_cci[10];
  char us_cdc[10];
  char us_objeto[10];
  char us_tp1[12];
  char us_tp2[12];
  char us_cdc1[10];

  int hn;

struct host_pto
          {
            char host[12];
        char puerto[5];
          }pto;

  if (argc != 6)
  {
   fprintf(stderr,"Uso: %s <usuario origen> <usuario destino> <atributo en MLS> <atributo en C> <atributo en
F>\n", argv[0]);
   exit(1);
  }


  strcpy(data.mls,argv[3]);
  strcpy(data.com,argv[4]);
```

```
  strcpy(data.fin,argv[5]);
  strcpy(data.us_origen,argv[1]);
  strcpy(data.us_destino,argv[2]);


/*  Localiza  host origen y destino  */

  if ((atrib = fopen("atrib.adm", "r+"))== NULL)
  {
    fprintf(stderr, "No se pudo abrir archivo.\n");
    return 1;
  }

for(k=1;k<=2;k++)
{
  fseek(atrib, 0, 0);
  fread(us_id, sizeof(us_id), 1, atrib);
  posicion=ftell(atrib);
  fseek(atrib, posicion, 0);
 while( strcmp(argv[k],us_id) && ! feof(atrib) )
  {
   posicion=ftell(atrib);
   fseek(atrib,posicion+86,0);
   if (feof(atrib))
       break;
   fread(us_id, sizeof(us_id), 1, atrib);
  }
if ( !strcmp(us_id,argv[k]) )
  {
   posicion=ftell(atrib);
   fseek(atrib,posicion,0);
   fread(us_host, sizeof(us_host), 1, atrib);
   posicion=ftell(atrib);
   fseek(atrib,posicion,0);
   fread(us_cat_mls, sizeof(us_cat_mls), 1, atrib);
   posicion=ftell(atrib);
   fseek(atrib,posicion,0);
   fread(us_clas_mls, sizeof(us_clas_mls), 1, atrib);
   posicion=ftell(atrib);
   fseek(atrib,posicion,0);
   fread(us_cci, sizeof(us_cci), 1, atrib);
   posicion=ftell(atrib);
   fseek(atrib,posicion,0);
   fread(us_cdc, sizeof(us_cdc), 1, atrib);
   posicion=ftell(atrib);
```

```
  fseek(atrib,posicion,0);
  fread(us_objeto, sizeof(us_objeto), 1, atrib);
  posicion=ftell(atrib);
  fseek(atrib,posicion,0);
  fread(us_tp1, sizeof(us_tp1), 1, atrib);
  posicion=ftell(atrib);
  fseek(atrib,posicion,0);
  fread(us_tp2, sizeof(us_tp2), 1, atrib);


  if (k==1)   /* origen   */
    {
     strcpy(data.host_origen,us_host);
          strcpy(data.us_o_cat_mls,us_cat_mls);
          strcpy(data.us_o_clas_mls,us_clas_mls);
          strcpy(data.us_o_cci,us_cci);
          strcpy(data.us_o_cdc,us_cdc);
     strcpy(us_cdc1,us_cdc);
          strcpy(data.us_o_objeto,us_objeto);
          strcpy(data.us_o_tp1,us_tp1);
     strcpy(data.us_o_tp2,us_tp2);
    }

  if (k==2)   /*  destino   */
        {
     strcpy(data.host_destino,us_host);
          strcpy(data.us_d_cat_mls,us_cat_mls);
          strcpy(data.us_d_clas_mls,us_clas_mls);
          strcpy(data.us_d_cci,"");
          strncat(data.us_d_cci,us_cci,10);
          strcpy(data.us_d_cdc,us_cdc);
          strcpy(data.us_d_objeto,us_objeto);
          strcpy(data.us_d_tp1,us_tp1);
     strcpy(data.us_d_tp2,us_tp2);
    }
 }
else
 {
  if(k==1)
    {
     printf("\n El usuario origen no existe \n");
     exit(1);
        }
  if(k==2)
   {
```

```
        printf("\n El usuario destino no existe \n");
        exit(1);
          }
    }

  } /*  fin del for  */
    fclose(atrib);

  strcpy(data.us_o_cdc,us_cdc1);
   if (strcmp(data.mls,"u") & strcmp(data.mls,"c") )
    if ( strcmp(data.mls,"s") && strcmp(data.mls,"t"))
     {
      printf("\n error en atributo para MLS (3er. parametro \n");
          exit(1);
     }


   if(strcmp(data.com,"udi") && strcmp(data.com,"cdi") )
     {
          printf("\n error en atributo para C (4o. parametro) \n");
          exit(1);
     }

   if(strcmp(data.fin,"s") && strcmp(data.fin,"n") )
     {
          printf("\n error en atributo para F (5o. parametro) \n");
          exit(1);
     }
  install_usr_handlers();

/*  Aqui se realiza la conexion con la AM    */

   portnumber = 3000;
  if ((outfd = u_connect(portnumber, "mexico")) < 0)
  {
   u_error("No se pudo establecer una conexion Internet");
   exit(1);
  }
  strcpy(buf,"");
  byteswritten = u_write(outfd, &data , sizeof(struct datag));
   if (byteswritten < 0)
   {
     fprintf(stderr, "Error escribiendo %s bytes, %ld bytes escritos\n",
           buf, (long)byteswritten);
   }
```

```
/* Recibe la respuesta de la AM    */

   while( (bytesread = u_read(outfd, buf, BLKSIZE)) > 1)
   {
    buf[80] = '\0';
    printf("\n  %s \n",buf);
   }
if( !strcmp(buf,"OK Transmision aprobada!"))
{
 /* Selecciona el puerto que se usara para la conexion del cliente
     con el servidor de mensajes
 */
   if ((puertos = fopen("puertos.adm", "r+"))== NULL)
   {
    fprintf(stderr, "No se pudo abrir archivo.\n");
    return 1;
   }
   fseek(puertos, 0, 0);
   fread(pto.host, sizeof(pto.host), 1, puertos);
   posicion=ftell(puertos);
   fseek(puertos, posicion, 0);
  while( strcmp(data.host_destino,pto.host) && ! feof(puertos) )
   {
    posicion=ftell(puertos);
    fseek(puertos,posicion+12,0);
    fread(pto.host, sizeof(pto.host), 1, puertos);
   }

  if ( !strcmp(pto.host,data.host_destino) )
   {
    posicion=ftell(puertos);
    fseek(puertos,posicion,0);
    fread(pto.puerto, sizeof(pto.puerto), 1, puertos);
    fclose(puertos);
   }

/*  Se ejecuta el programa que utilizara el cliente para el envio de
    mensajes    */

   execlp("client1", "client1", data.host_destino,pto.puerto, (char *) 0);
}
 u_close(outfd);
 exit(0);
}
```

```
/*
 programa : servidor.c  Servidor de mensajes
 autor    : Salvador Contreras H.
 fecha    : Diciembre de 1997
 objetivo : Recibir los mensajes en el host destino y evaluar
            las politicas locales
*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <limits.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/uio.h>
#include "uici.h"

#define BLKSIZE 1024
#define MAX_CANON 1024

usr1handler(s)
 int s;
{
 fprintf(stderr,"SIGUR1 \n");
}
usr2handler(s)
 int s;
{
 fprintf(stderr,"SIGUR2 \n");
}
install_usr_handlers()
{
 struct sigaction newact;
 newact.sa_handler = usr1handler();
 sigemptyset(&newact.sa_mask);
 newact.sa_flags = 0;
 if (sigaction(SIGUSR1, &newact, (struct sigaction *) NULL) == -1)
 {
  perror(" No se pudo instalar el handler de la senal SIGUR1 ");
  return;
 }
 newact.sa_handler = usr2handler();
 if (sigaction(SIGUSR2, &newact, (struct sigaction *) NULL) == -1)
 {
  perror(" No se pudo instalar el handler de la senal SIGUR2 ");
```

```
  return;
 }
 fprintf(stderr,"Proceso servidor %ld listo \n",
    (long)getpid());
}

 struct datag
   {
    int  pol_o_mls;
    int  pol_d_mls;
    int  pol_o_com;
    int  pol_d_com;
    int  pol_o_fin;
    int  pol_d_fin;
    char us_origen[10];
    char us_destino[10];
    char host_origen[12];
    char host_destino[12];
    char mls[10];
    char com[10];
    char fin[10];
    char cci[10];
    char cdc[10];
    char us_o_cat_mls[3];
    char us_d_cat_mls[3];
    char us_o_clas_mls[4];
    char us_d_clas_mls[4];
    char us_o_cci[10];
    char us_d_cci[10];
    char us_o_cdc[10];
    char us_d_cdc[10];
    char us_o_objeto[10];
    char us_d_objeto[10];
    char us_o_tp1[12];
    char us_d_tp1[12];
    char us_o_tp2[12];
    char us_d_tp2[12];
   }data;
```

```
main(argc, argv)
 int argc;
 char *argv[];
{

  unsigned short portnumber;
  int listenfd;
  int communfd;
  size_t bytesread;
  size_t byteswritten;
  char buf[BLKSIZE];
  char remote[MAX_CANON];
  char respuesta[80];
  int evalua_mls;
  int evalua_com;
  int evalua_fin;

  if (argc != 2)
  {
    fprintf(stderr,"Uso: %s puerto \n", argv[0]);
    exit(1);
  }
 install_usr_handlers();
 portnumber = (unsigned short) atoi(argv[1]);
 if ((listenfd = u_open(portnumber)) < 0)
 {
  u_error("No se pudo establecer un puerto de conexion\n");
  exit(1);
 }
for(;;)
{
 if ((communfd = u_listen(listenfd, remote)) < 0)
 {
  u_error("Falla en el intento de escuchar \n");
  exit(1);
 }
 fprintf(stderr,"La conexion ha sido realizada a %s\n", argv[1]);
 bytesread=u_read(communfd, &data, sizeof(struct datag));
 evalua_mls=pol_mls();
 evalua_com=pol_com();
 evalua_fin=pol_fin();
```

```
 /*  Envia respuesta a la AM  */

if (evalua_mls==1 &&  evalua_com==1 && evalua_fin ==1 )
 strcpy(respuesta,"OK Transmision aprobada!");
else
 strcpy(respuesta,"Transmision negada!");
 strcpy(buf,respuesta);
 buf[80] = '\0';
 byteswritten = u_write(communfd,buf, 1024);
 if (byteswritten < 0)
  printf("Error en el envio de respuesta \n");
 printf("\n Respuesta enviada \n ");
if ( !strcmp(respuesta,"OK Transmision aprobada!"))

{
  if ((communfd = u_listen(listenfd, remote)) < 0)
   {
    u_error("Falla en el intento de escuchar \n");
    exit(1);
   }
   fprintf(stderr,"La conexion ha sido realizada a %s\n", argv[1]);
   while( (bytesread = u_read(communfd, buf, BLKSIZE)) > 1)
    {
    byteswritten = write(STDOUT_FILENO, buf, bytesread);
    if (bytesread != byteswritten)
     {
       fprintf(stderr, "Error escribiendo %d bytes, %ld bytes escritos\n",
           (long)bytesread, (long)byteswritten);
       break;
     }
    }
 }
}
 u_close(listenfd);
 u_close(communfd);
 exit(0);
}
```

```
/*  Politicas Locales  */


int pol_mls()
{
   int us_cl,info_cl;
   char *us_d_clas;
   char *info_clas;
   int  cl;
   FILE *usa;


   us_d_clas=data.us_d_clas_mls;
   switch(*us_d_clas)
    {
     case 'u': us_cl=0;
                    break;
     case 'c': us_cl=1;
                    break;
     case 's': us_cl=2;
                    break;
     case 't': us_cl=3;
    }

   info_clas=data.mls;
   switch(*info_clas)
    {
     case 'u': info_cl=0;
                    break;
     case 'c': info_cl=1;
                    break;
     case 's': info_cl=2;
                    break;
     case 't': info_cl=3;
   }


/* Categoria del usuario destino y de la informacion */

      if (us_cl >= info_cl)
      {
           printf("\n Politica Multinivel aprobada");
       return 1;
      }
     else
```

```
      {
          printf("\n Politica Multinivel no aprobada");
       return 0;
      }
printf("\n clas de usuario : %d",us_cl);
printf("\n clas de la info : %d", info_cl);
}

int pol_com()
{
 if (!strcmp(data.us_d_objeto,"mensaje"))
  if(!strcmp(data.us_d_tp2, "servidor") || !strcmp(data.us_d_tp1,"servidor"))
          {
            printf("\n Politica Comercial aprobada ");
            return 1;
          }
   else
     {
          printf("\n Politica Comercial no aprobada ");
       return 0;
          }
}


int pol_fin()
{
  FILE *ap_cci_o;
  FILE *ap_cci_d;
  char cci_vi[10];
  int posicion;
  int e_fin;

  if (!strncmp(data.us_d_cdc, data.us_o_cdc,10) || !strcmp(data.fin,"s") )
    {
      printf("\n Politica Financiera aprobada ");
      e_fin=1;
    }
   else

  {
    if ((ap_cci_o = fopen(data.us_origen, "a+"))== NULL)
          {
           fprintf(stderr, "Imposible abrir archivo.\n");
          return 1;
          }
```

```
            fseek(ap_cci_o, 0, 0);
            fread(cci_vi, sizeof(cci_vi), 1, ap_cci_o);
            posicion=ftell(ap_cci_o);
            fseek(ap_cci_o, posicion, 0);
           while( strncmp(cci_vi,data.us_o_cci,10) && ! feof(ap_cci_o) )
            {
              posicion=ftell(ap_cci_o);
              fseek(ap_cci_o,posicion+10,0);
              fread(cci_vi, sizeof(cci_vi), 1, ap_cci_o);
            }

      if ( strncmp(cci_vi,data.us_d_cci,10) )
           {
        printf("\n Politica Financiera aprobada ");
        if ((ap_cci_d = fopen(data.us_destino, "a+"))== NULL)
                  {
                    fprintf(stderr, "Imposible abrir archivo.\n");
                 return 1;
                }
        fwrite(data.us_o_cci, sizeof(data.us_o_cci), 1, ap_cci_d);
        fclose(ap_cci_d);
        e_fin=1;
       }
      else
       {
            printf("\n Politica Financiera no aprobada ");
            e_fin=0;
       }
      fclose(ap_cci_o);
    }  /* fin del else  */

 if (e_fin == 1)
   return 1;
 else
   return 0;

}
```

/* programa para envío de mensajes desde el origen */

cliente.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <errno.h>
#include <sys/types.h>
#include "uici.h"

#define BLKSIZE 1024
#define MAX_CANON 1024

usr1handler(s)
 int s;
{
 fprintf(stderr,"Senal SIGUR1 recibida \n");
}
usr2handler(s)
 int s;
{
 fprintf(stderr,"Senal SIGUR2 recibida \n");
}
install_usr_handlers()
{
 struct sigaction newact;
 newact.sa_handler = usr1handler;
 sigemptyset(&newact.sa_mask);
 newact.sa_flags = 0;
 if (sigaction(SIGUSR1, &newact, (struct sigaction *) NULL) == -1) {
  perror(" No se pudo instalar el handler de la senal SIGUR1 ");
  return;
 }
 newact.sa_handler = usr2handler;
 if (sigaction(SIGUSR2, &newact, (struct sigaction *) NULL) == -1) {
  perror(" No se pudo instalar el handler de la senal SIGUR2 ");
  return;
 }
 fprintf(stderr,"Proceso cliente %ld listo para usar SIGUSR1 y SIGUSR2 \n",
    (long)getpid());
}




main(argc, argv)
 int argc;
 char *argv[];
{
  unsigned short portnumber;
  int outfd;
  size_t bytesread;
  size_t byteswritten;
```

```
  char buf[BLKSIZE];
  size_t l_atributos;
  int i;
  int communfd, listenfd;
  char remote[MAX_CANON];
  if (argc != 3)
  {
   fprintf(stderr,"Uso: %s host puerto <atributos de politicas>\n", argv[0]);
   exit(1);
  }

install_usr_handlers();
printf("\n argv[1] %s", argv[1]);
printf("\n argv[2] %s", argv[2]);
portnumber = (unsigned short) atoi(argv[2]);
 if ((outfd = u_connect(portnumber, argv[1])) < 0) {
  u_error("Incapaz de establecer una conexion Internet");
  exit(1);
 }

 fprintf(stderr,"La conexion ha sido realizada a %s\n", argv[1]);
 strcpy(buf,"");

for( ; ;)
 {
  bytesread = read(STDIN_FILENO, buf, BLKSIZE);
  if  ((bytesread == -1) && (errno = EINTR) )
    fprintf(stderr,"Cliente reestableciendo lectura \n");
  else
    if (bytesread <= 0)
     break;
    else
      if (bytesread==1)
       {
        printf("\n Fin...\n");
        break;
       }


    else {
          byteswritten = u_write(outfd, buf, bytesread);
             if (byteswritten != bytesread)
           {
            fprintf(stderr, "Error escribiendo %d bytes, %ld bytes escritos\n",
            (long)bytesread, (long)byteswritten);
             break;
           }
  }
 }

 u_close(outfd);
 exit(0);
}
```

/* Programa para ingresar usuarios - atributos */

atrins.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
  FILE *atrib;
  int posicion;
  char us_id[10];
  char us[10];
  char us_host[12];
  char us_cat_mls[10];
  char us_clas_mls[10];
  char us_cci[10];
  char us_cdc[10];
  char us_objeto[10];
  char us_tp1[12];
  char us_tp2[12];


  if ((atrib = fopen("atrib.adm", "a+"))== NULL)
  {
    fprintf(stderr, "Imposible abrir archivo.\n");
    return 1;
  }
     printf("\nidentificador de usuario :");
     gets(us);
     fseek(atrib, 0, 0);
     fread(us_id, sizeof(us_id), 1, atrib);
     while( strcmp(us,us_id) && ! feof(atrib) )
      {
       posicion=ftell(atrib);
       fseek(atrib,posicion+86,0);
       fread(us_id, sizeof(us_id), 1, atrib);
      }

    if( !strcmp(us_id, us) )
      {
       printf("\n usuario %s ya existe \n",us);
       fclose(atrib);
       exit(1);
      }


   if ( strcmp(us_id, us) )
     {
       printf("\nhost: ");
       gets(us_host);
       printf("\ncategoria en mls :");
       gets(us_cat_mls);
       printf("\nclasificacion en mls :");
```

```
    gets(us_clas_mls);
    printf("\nclase de conflicto de interes :");
    gets(us_cci);
    printf("\nconjunto de datos de cia. :");
    gets(us_cdc);
    printf("\nobjeto :");
    gets(us_objeto);
    printf("\ntp1 :");
    gets(us_tp1);
    printf("\ntp2 :");
    gets(us_tp2);

    fwrite(us, sizeof(us), 1, atrib);
    fwrite(us_host, sizeof(us_host), 1, atrib);
    fwrite(us_cat_mls, sizeof(us_cat_mls), 1, atrib);
    fwrite(us_clas_mls, sizeof(us_clas_mls), 1, atrib);
    fwrite(us_cci, sizeof(us_cci), 1, atrib);
    fwrite(us_cdc, sizeof(us_cdc), 1, atrib);
    fwrite(us_objeto, sizeof(us_objeto), 1, atrib);
    fwrite(us_tp1, sizeof(us_tp1), 1, atrib);
    fwrite(us_tp2, sizeof(us_tp2), 1, atrib);
   }
 fclose(atrib);
 printf("\n");
 return 0;
}
```

/* programa para consultas de usuarios - atributos */

atrcon.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  FILE *atrib;
  int posicion;
  char us_id[10];
  char us_host[12];
  char us_cat_mls[10];
  char us_clas_mls[10];
  char us_cci[10];
  char us_cdc[10];
  char us_objeto[10];
  char us_tp1[12];
  char us_tp2[12];

 if (argc!= 2)
   {
    printf("\numero incorrecto de parametros \n");
    exit(1);
   }
 if ((atrib = fopen("atrib.adm", "r"))== NULL)
 {
   fprintf(stderr, "Imposible abrir archivo.\n");
   return 1;
 }
 fseek(atrib, 0, 0);
 fread(us_id, sizeof(us_id), 1, atrib);
while( strcmp(argv[1],us_id) && ! feof(atrib) )
 {
  posicion=ftell(atrib);
  fseek(atrib,posicion+86,0);
  fread(us_id, sizeof(us_id), 1, atrib);
 }
if( strcmp(us_id, argv[1]) )
 {
  printf("\n usuario %s no existe \n",argv[1]);
  fclose(atrib);
  exit(1);
 }
if ( !strcmp(us_id, argv[1]) )
 {
  posicion=ftell(atrib);
  fseek(atrib,posicion,0);
  fread(us_host, sizeof(us_host), 1, atrib);
  posicion=ftell(atrib);
  fseek(atrib,posicion,0);
  fread(us_cat_mls, sizeof(us_cat_mls), 1, atrib);

  posicion=ftell(atrib);
  fseek(atrib,posicion,0);
  fread(us_clas_mls, sizeof(us_clas_mls), 1, atrib);
```

```
    posicion=ftell(atrib);
    fseek(atrib,posicion,0);
    fread(us_cci, sizeof(us_cci), 1, atrib);

    posicion=ftell(atrib);
    fseek(atrib,posicion,0);
    fread(us_cdc, sizeof(us_cdc), 1, atrib);

    posicion=ftell(atrib);
    fseek(atrib,posicion,0);
    fread(us_objeto, sizeof(us_objeto), 1, atrib);

    posicion=ftell(atrib);
    fseek(atrib,posicion,0);
    fread(us_tp1, sizeof(us_tp1), 1, atrib);

    posicion=ftell(atrib);
    fseek(atrib,posicion,0);
    fread(us_tp2, sizeof(us_tp2), 1, atrib);

    printf(" usuario: %s",us_id);
    printf("\n host : %s",us_host);
    printf("\n categoria en mls: %s",us_cat_mls);
    printf("\n clasificacion en mls: %s",us_clas_mls);
    printf("\n clase de conflicto de interes: %s",us_cci);
    printf("\n conjunto de datos de cia.: %s",us_cdc);
    printf("\n objeto: %s",us_objeto);
    printf("\n tp1: %s",us_tp1);
    printf("\n tp2: %s\n",us_tp2);
  }
    fclose(atrib);
    return 0;
}
```

/* programa para cambios de usuarios - atributos */

atrcam.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  FILE *atrib;
  int posicion;
  char us_id[10];
  char us_host[12];
  char us_cat_mls[10];
  char us_clas_mls[10];
  char us_cci[10];
  char us_cdc[10];
  char us_objeto[10];
  char us_tp1[12];
  char us_tp2[12];

  char us_id_nuevo[10];

 if (argc!= 3)
   {
    printf("\n numero incorrecto de parametros \n");
    exit(1);
   }

 if ((atrib = fopen("atrib.adm", "r+b"))== NULL)
 {
    fprintf(stderr, "Imposible abrir archivo.\n");
    return 1;
 }

 fseek(atrib, 0, 0);
 fread(us_id, sizeof(us_id), 1, atrib);
while( strcmp(argv[1],us_id) && ! feof(atrib) )
 {
  posicion=ftell(atrib);
  fseek(atrib,posicion+86,0);
  fread(us_id, sizeof(us_id), 1, atrib);
  if (feof(atrib))
        break;
 }



if (strcmp(us_id,argv[1]))
  {
   printf("\n usuario no existe \n");
   fclose(atrib);
   exit(1);
  }
  if(!strcmp(argv[2],"u"))
   {
        printf("\n nuevo identificador -> ");
```

```
        gets(us_id_nuevo);
        fseek(atrib, 0, 0);
        fread(us_id, sizeof(us_id), 1, atrib);
        while( strcmp(us_id_nuevo,us_id) && ! feof(atrib) )
         {
          posicion=ftell(atrib);
          fseek(atrib,posicion+86,0);
          fread(us_id, sizeof(us_id), 1, atrib);
         }
        if( !strcmp(us_id,us_id_nuevo) )
         {
          printf("\n identificador de usuario ya existe ");
          fclose(atrib);
          exit(1);
         }

        if (!strcmp(us_id_nuevo,""))
         {
          printf("\n identificador de usuario no valido \n");
     fclose(atrib);
          exit(1);
         }
          posicion=ftell(atrib);
          fseek(atrib,posicion-10,0);
          fwrite(us_id_nuevo, sizeof(us_id_nuevo), 1, atrib);
          printf(" usuario: %s",us_id_nuevo);
      }
if(!strcmp(argv[2],"h"))
{
 printf("\n nuevo host -> ");
 gets(us_host);
 if (strcmp(us_host,""))
       {
        posicion=ftell(atrib);
        fseek(atrib, posicion, 0);
        fwrite(us_host, sizeof(us_host), 1, atrib);
       }
}
if(!strcmp(argv[2],"m"))
{
 printf("\n nueva categoria en mls  -> ");
 gets(us_cat_mls);
 printf("\n nueva clasificacion en mls ->");
 gets(us_clas_mls);
 if (!strcmp(us_cat_mls,""))
       strcpy(us_cat_mls,"0");

 posicion=ftell(atrib);
 fseek(atrib, posicion+12, 0);
 fwrite(us_cat_mls, sizeof(us_cat_mls), 1, atrib);
 if (!strcmp(us_clas_mls,""))
       strcpy(us_clas_mls,"0");
 posicion=ftell(atrib);
 fseek(atrib, posicion, 0);
 fwrite(us_clas_mls, sizeof(us_clas_mls), 1, atrib);
}
```

```
    if(!strcmp(argv[2],"f"))
    {
     printf("\n nueva cci  -> ");
     gets(us_cci);
     printf("\n nuevo cdc ->");
     gets(us_cdc);

     if (!strcmp(us_cci,""))
           strcpy(us_cci,"0");

     posicion=ftell(atrib);
     fseek(atrib, posicion+32, 0);
     fwrite(us_cci, sizeof(us_cci), 1, atrib);

     if (!strcmp(us_cdc,""))
           strcpy(us_cdc,"0");

     posicion=ftell(atrib);
     fseek(atrib, posicion, 0);
     fwrite(us_cdc, sizeof(us_cdc), 1, atrib);
    }

   if(!strcmp(argv[2],"c"))
   {
    printf("\n nuevo objeto   -> ");
    gets(us_objeto);
    printf("\n nuevo tp1 ->");
    gets(us_tp1);
    printf("\n nuevo tp2 ->");
    gets(us_tp2);
    if (!strcmp(us_objeto,""))
          strcpy(us_objeto,"0");

    posicion=ftell(atrib);
    fseek(atrib, posicion+52, 0);
    fwrite(us_objeto, sizeof(us_objeto), 1, atrib);

    if (!strcmp(us_tp1,""))
          strcpy(us_tp1,"0");

    posicion=ftell(atrib);
    fseek(atrib,posicion,0);
    fwrite(us_tp1, sizeof(us_tp1), 1, atrib);

    if (!strcmp(us_tp2,""))
          strcpy(us_tp2,"0");

    posicion=ftell(atrib);
    fseek(atrib,posicion,0);
    fwrite(us_tp2, sizeof(us_tp2), 1, atrib);
   }
   fclose(atrib);
   return 0;
}
```

/* programa para eliminar usuarios - atributos */

atreli.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  FILE *atrib, *temp;
  int posicion;
  char us[10];
  char resp[2];

  char us_id[10];
  char us_host[12];
  char us_cat_mls[10];
  char us_clas_mls[10];
  char us_cci[10];
  char us_cdc[10];
  char us_objeto[10];
  char us_tp1[12];
  char us_tp2[12];


 if (argc!= 2)
  {
   printf("\n numero incorrecto de parametros \n");
   exit(1);
  }

 if ((atrib = fopen("atrib.adm", "r"))== NULL)
 {
   fprintf(stderr, "Imposible abrir archivo.\n");
   return 1;
 }

if ((temp = fopen("temp.adm", "w+"))== NULL)
 {
   fprintf(stderr, "Imposible abrir archivo.\n");
   return 1;
 }


 fseek(atrib, 0, 0);
 fread(us_id, sizeof(us_id), 1, atrib);
while( strcmp(argv[1],us_id) && ! feof(atrib) )
 {
  posicion=ftell(atrib);
  fseek(atrib,posicion+86,0);
  fread(us_id, sizeof(us_id), 1, atrib);
 }

 if( strcmp(us_id,argv[1]) )
{
 printf("\n usuario %s no existe \n",argv[1]);
 fclose(atrib);
```

```
  fclose(temp);
  exit(1);
 }

if ( !strcmp(us_id,argv[1]) )
 do
  {
    printf("\n Desea borrar %s ? ",argv[1]);
    scanf("%s",resp);
  }while(strcmp(resp,"s")&&strcmp(resp,"S")&&strcmp(resp,"n")&&strcmp(resp,"N"));

  if( !strcmp(resp,"n") || !strcmp(resp,"N"))
   {
   fclose(atrib);
   fclose(temp);
   exit(1);
   }

  strcpy(us,"");
  fseek(atrib,0,0);
  while( ! feof(atrib) )
  {
   posicion=ftell(atrib);
   fseek(atrib,posicion,0);
   fread(us_id, sizeof(us_id), 1, atrib);

   posicion=ftell(atrib);
   fseek(atrib,posicion,0);
   fread(us_host, sizeof(us_host), 1, atrib);

   posicion=ftell(atrib);
   fseek(atrib,posicion,0);
   fread(us_cat_mls, sizeof(us_cat_mls), 1, atrib);

   posicion=ftell(atrib);
   fseek(atrib,posicion,0);
   fread(us_clas_mls, sizeof(us_clas_mls), 1, atrib);

   posicion=ftell(atrib);
   fseek(atrib,posicion,0);
   fread(us_cci, sizeof(us_cci), 1, atrib);

   posicion=ftell(atrib);
   fseek(atrib,posicion,0);
   fread(us_cdc, sizeof(us_cdc), 1, atrib);

   posicion=ftell(atrib);
   fseek(atrib,posicion,0);
   fread(us_objeto, sizeof(us_objeto), 1, atrib);

   posicion=ftell(atrib);
   fseek(atrib,posicion,0);
   fread(us_tp1, sizeof(us_tp1), 1, atrib);

   posicion=ftell(atrib);
   fseek(atrib,posicion,0);
   fread(us_tp2, sizeof(us_tp2), 1, atrib);
```

```
 if( strcmp(argv[1],us_id) && strcmp(us_id,us) )
   {
        fwrite(us_id, sizeof(us_id), 1, temp);
        fwrite(us_host, sizeof(us_host), 1, temp);
        fwrite(us_cat_mls, sizeof(us_cat_mls), 1, temp);
        fwrite(us_clas_mls, sizeof(us_clas_mls), 1, temp);
        fwrite(us_cci, sizeof(us_cci), 1, temp);
        fwrite(us_cdc, sizeof(us_cdc), 1, temp);
        fwrite(us_objeto, sizeof(us_objeto), 1, temp);
        fwrite(us_tp1, sizeof(us_tp1), 1, temp);
        fwrite(us_tp2, sizeof(us_tp2), 1, temp);
   }
  strcpy(us,us_id);
}

  fclose(atrib);
  fclose(temp);


if ((atrib = fopen("atrib.adm", "w+"))== NULL)
 {
   fprintf(stderr, "Imposible abrir archivo.\n");
   return 1;
 }

if ((temp = fopen("temp.adm", "r+"))== NULL)
 {
   fprintf(stderr, "Imposible abrir archivo.\n");
   return 1;
 }
strcpy(us_id,"");
strcpy(us_host,"");
strcpy(us_cat_mls,"");
strcpy(us_clas_mls,"");
strcpy(us_cci,"");
strcpy(us_cdc,"");
strcpy(us_objeto,"");
strcpy(us_tp1,"");
strcpy(us_tp2,"");

fseek(temp,0,0);
 while( ! feof(temp) )
 {
  posicion=ftell(temp);
  fseek(temp,posicion,0);
  fread(us_id, sizeof(us_id), 1, temp);
  posicion=ftell(temp);
  fseek(temp,posicion,0);
  fread(us_host, sizeof(us_host), 1, temp);
  posicion=ftell(temp);
  fseek(temp,posicion,0);
  fread(us_cat_mls, sizeof(us_cat_mls), 1, temp);
  posicion=ftell(temp);
  fseek(temp,posicion,0);
  fread(us_clas_mls, sizeof(us_clas_mls), 1, temp);
  posicion=ftell(temp);
  fseek(temp,posicion,0);
```

```
    fread(us_cci, sizeof(us_cci), 1, temp);
    posicion=ftell(temp);
    fseek(temp,posicion,0);
    fread(us_cdc, sizeof(us_cdc), 1, temp);
    posicion=ftell(temp);
    fseek(temp,posicion,0);
    fread(us_objeto, sizeof(us_objeto), 1, temp);
    posicion=ftell(temp);
    fseek(temp,posicion,0);
    fread(us_tp1, sizeof(us_tp1), 1, temp);
    posicion=ftell(temp);
    fseek(temp,posicion,0);
    fread(us_tp2, sizeof(us_tp2), 1, temp);


    if(strcmp(us,us_id))
     {
          fwrite(us_id, sizeof(us_id), 1, atrib);
          fwrite(us_host, sizeof(us_host), 1, atrib);
          fwrite(us_cat_mls, sizeof(us_cat_mls), 1, atrib);
          fwrite(us_clas_mls, sizeof(us_clas_mls), 1, atrib);
          fwrite(us_cci, sizeof(us_cci), 1, atrib);
          fwrite(us_cdc, sizeof(us_cdc), 1, atrib);
          fwrite(us_objeto, sizeof(us_objeto), 1, atrib);
          fwrite(us_tp1, sizeof(us_tp1), 1, atrib);
          fwrite(us_tp2, sizeof(us_tp2), 1, atrib);
     }
    strcpy(us,us_id);
  }

    fclose(atrib);
    fclose(temp);

    return 0;
}
```

/* programa para ingresar hosts */

hostins.c

```
#include <stdio.h>

struct host
        {
          char host[12];
          char mls[2];
          char com[2];
          char fin[2];
        }h_pol,buf;

int main(int argc, char *argv[])
{
 FILE *hosts;
 int posicion;

 if (argc!=5)
   {
        printf("\n numero incorrecto de parametros");
        exit(1);
   }
 if ((hosts = fopen("politica.adm", "a+"))== NULL)
 {
   fprintf(stderr, "Imposible abrir archivo.\n");
   return 1;
 }
 fseek(hosts, 0, 0);
 fread(h_pol.host, sizeof(h_pol.host), 1, hosts);
 posicion=ftell(hosts);
 fseek(hosts, posicion, 0);
while( strcmp(argv[1],h_pol.host) && ! feof(hosts) )
 {
  posicion=ftell(hosts);
  fseek(hosts,posicion+6,0);
  fread(h_pol.host, sizeof(h_pol.host), 1, hosts);
 }
if ( !strcmp(h_pol.host,argv[1]) )
 {
  printf("\n host %s ya existe \n",h_pol.host);
  fclose(hosts);
  exit(1);
 }
    strcpy(h_pol.host,argv[1]);
    strcpy(h_pol.mls,argv[2]);
    strcpy(h_pol.com,argv[3]);
    strcpy(h_pol.fin,argv[4]);
    fwrite(h_pol.host, sizeof(h_pol.host), 1, hosts);
    fwrite(h_pol.mls, sizeof(h_pol.mls), 1, hosts);
    fwrite(h_pol.com, sizeof(h_pol.com), 1, hosts);
    fwrite(h_pol.fin, sizeof(h_pol.fin), 1, hosts);
    return 0;
 }
```

/* programa para consultar políticas de un host */

hostcon.c

```
#include <stdio.h>

struct host
        {
          char host[12];
          char mls[2];
          char com[2];
          char fin[2];
        }h_pol,buf;

int main(int argc, char *argv[])
{
  FILE *hosts;
  int posicion;

 if (argc!= 2)
  {
   printf("\numero incorrecto de parametros ");
   exit(1);
  }

 if ((hosts = fopen("politica.adm", "r+"))== NULL)
 {
   fprintf(stderr, "Imposible abrir archivo.\n");
   return 1;
 }

 fseek(hosts, 0, 0);
 fread(h_pol.host, sizeof(h_pol.host), 1, hosts);
 posicion=ftell(hosts);
 fseek(hosts, posicion, 0);
while( strcmp(argv[1],h_pol.host) && ! feof(hosts) )
 {
  posicion=ftell(hosts);
  fseek(hosts,posicion+6,0);
  fread(h_pol.host, sizeof(h_pol.host), 1, hosts);
 }


if( feof(hosts) &&  strcmp(h_pol.host,argv[1]))
{
 printf("\n host %s no existe \n",argv[1]);
 fclose(hosts);
 exit(1);
}


if ( !strcmp(h_pol.host,argv[1]) )
 {
  printf("\n host: %s",h_pol.host);
  posicion=ftell(hosts);
  fseek(hosts,posicion,0);
```

```
    fread(h_pol.mls, sizeof(h_pol.mls), 1, hosts);
    printf("\n mls: %s",h_pol.mls);
    posicion=ftell(hosts);
    fseek(hosts,posicion,0);
    fread(h_pol.com, sizeof(h_pol.com), 1, hosts);
    printf("\n com: %s",h_pol.com);
    posicion=ftell(hosts);
    fseek(hosts,posicion,0);
    fread(h_pol.fin, sizeof(h_pol.fin), 1, hosts);
    printf("\n fin: %s\n",h_pol.fin);
    fclose(hosts);
   }
   return 0;
}
```

/* programa para cambiar políticas en host */

hostcam.c

```
#include <stdio.h>

  FILE * ap;
  int posicion,pos_act,parametros;
  char host[12];
  char mls[2], com[2], fin[2];

int main(int argc, char *argv[])
{
  parametros=argc;
  if(argc<3)
    {
     printf("\n numero incorrecto de parametros\n");
     exit(1);
    }
  if (argc>5)
    {
          printf("\n numero incorrecto de parametros\n");
          exit(1);
    }

if(argc==3)
 {
 if( strcmp(argv[2],"m") && strcmp(argv[2],"c") && strcmp(argv[2],"f") )
 if( strcmp(argv[2],"-m") && strcmp(argv[2],"-c") && strcmp(argv[2],"-f") )
   {
    printf("\n parametro %s no valido \n",argv[2]);
    exit(1);
   }
 }

if(argc==4)
 {
 if( strcmp(argv[2],"m") && strcmp(argv[2],"c") && strcmp(argv[2],"f") )
 if( strcmp(argv[2],"-m") && strcmp(argv[2],"-c") && strcmp(argv[2],"-f") )
   {
    printf("\n parametro %s no valido \n",argv[2]);
    exit(1);
   }
 if( strcmp(argv[3],"m") && strcmp(argv[3],"c") && strcmp(argv[3],"f") )
 if( strcmp(argv[3],"-m") && strcmp(argv[3],"-c") && strcmp(argv[3],"-f") )
   {
    printf("\n parametro %s no valido \n",argv[3]);
    exit(1);
   }
 }

if(argc==5)
 {
 if( strcmp(argv[2],"m") && strcmp(argv[2],"c") && strcmp(argv[2],"f") )
 if( strcmp(argv[2],"-m") && strcmp(argv[2],"-c") && strcmp(argv[2],"-f") )
  {
```

```
    printf("\n parametro %s no valido \n",argv[2]);
    exit(1);
   }
  if( strcmp(argv[3],"m") && strcmp(argv[3],"c") && strcmp(argv[3],"f") )
  if( strcmp(argv[3],"-m") && strcmp(argv[3],"-c") && strcmp(argv[3],"-f") )
   {
    printf("\n parametro %s no valido \n",argv[3]);
    exit(1);
   }
  if( strcmp(argv[4],"m") && strcmp(argv[4],"c") && strcmp(argv[4],"f") )
  if( strcmp(argv[4],"-m") && strcmp(argv[4],"-c") && strcmp(argv[4],"-f") )
   {
    printf("\n parametro %s no valido \n",argv[4]);
    exit(1);
   }
 }


  if ((ap = fopen("politica.adm", "r+b"))== NULL)
  {
    fprintf(stderr, "Imposible abrir archivo.\n");
    return 1;
  }


  fseek(ap, 0, 0);
  fread(host, sizeof(host), 1, ap);
  posicion=ftell(ap);
  fseek(ap, posicion, 0);
 while( strcmp(argv[1],host) && ! feof(ap) )
  {
   posicion=ftell(ap);
   posicion=posicion+6;
   fseek(ap,posicion,0);
   fread(host, sizeof(host), 1, ap);
  }

pos_act=posicion+12;
if ( strcmp(host,argv[1]) )
  {
   printf("\n host %s no existe\n",argv[1]);
   fclose(ap);
   exit(1);
  }
    strcpy(host,argv[1]);
switch(argc)
{
 case 3:
         if( !strcmp(argv[2],"m") )
          {
            incluye_mls();
            break;
          }
         if( !strcmp(argv[2],"-m"))
          {
            elimina_mls();
            break;
```

```
        }
        if( !strcmp(argv[2],"c") )
        {
          incluye_com();
          break;
        }
        if( !strcmp(argv[2],"-c") )
        {
          elimina_com();
          break;
        }

        if( !strcmp(argv[2],"f") )
        {
          incluye_fin();
          break;
        }
        if( !strcmp(argv[2],"-f") )
        {
          elimina_fin();
          break;
        }

case 4:
        if( !strcmp(argv[2],"m") || !strcmp(argv[3],"m") )
          incluye_mls();
        if( !strcmp(argv[2],"-m") || !strcmp(argv[3],"-m") )
          elimina_mls();

        if( !strcmp(argv[2],"c") || !strcmp(argv[3],"c") )
          incluye_com();
        if( !strcmp(argv[2],"-c") || !strcmp(argv[3],"-c") )
          elimina_com();

        if( !strcmp(argv[2],"f") || !strcmp(argv[3],"f") )
          incluye_fin();
        if( !strcmp(argv[2],"-f") || !strcmp(argv[3],"-f") )
          elimina_fin();

case 5:
 if( !strcmp(argv[2],"m") || !strcmp(argv[3],"m") || !strcmp(argv[4],"m") )
          incluye_mls();
 if( !strcmp(argv[2],"-m") || !strcmp(argv[3],"-m") || !strcmp(argv[4],"-m") )
          elimina_mls();

if( !strcmp(argv[2],"c") || !strcmp(argv[3],"c") || !strcmp(argv[4],"c") )
          incluye_com();
if( !strcmp(argv[2],"-c") || !strcmp(argv[3],"-c") || !strcmp(argv[4],"-c") )
          elimina_com();

if( !strcmp(argv[2],"f") || !strcmp(argv[3],"f") || !strcmp(argv[4],"f") )
          incluye_fin();
if( !strcmp(argv[2],"-f") || !strcmp(argv[3],"-f") || !strcmp(argv[4],"-f") )
          elimina_fin();
}
fclose(ap);
return 0;
```

```
}
int incluye_mls()
 {
   strcpy(mls,"1");
   fseek(ap,pos_act,0);
   fwrite(mls, sizeof(mls),1,ap);
   return 1;
 }

int elimina_mls()
 {
   strcpy(mls,"0");
   fseek(ap,pos_act,0);
   fwrite(mls, sizeof(mls),1,ap);
   return 1;
 }

int incluye_com()
 {
   strcpy(com,"1");
   fseek(ap,pos_act+2,0);
   fwrite(com, sizeof(com),1,ap);
   return 1;
 }

int elimina_com()
 {
  strcpy(com,"0");
  fseek(ap,pos_act+2,0);
  fwrite(com, sizeof(com),1,ap);
  return 1;
 }

int incluye_fin()
 {
  strcpy(fin,"1");
  fseek(ap,pos_act+4,0);
  fwrite(fin, sizeof(fin),1,ap);
  return 1;
 }

int elimina_fin()
 {
  strcpy(fin,"0");
  fseek(ap,pos_act+4,0);
  fwrite(fin, sizeof(fin),1,ap);
  return 1;
 }
```

*/ programa para eliminar hosts */

hosteli.c

```
#include <stdio.h>

struct host
        {
          char host[12];
          char mls[2];
          char com[2];
          char fin[2];
        }h_pol,buf;

int main(int argc, char *argv[])
{
  FILE *hosts, *temp;
  char host[12];
  int posicion;
  char resp[2];

 if (argc!= 2)
   {
    printf("\numero incorrecto de parametros ");
    exit(1);
   }

 if ((hosts = fopen("politica.adm", "r+"))== NULL)
  {
    fprintf(stderr, "Imposible abrir archivo.\n");
    return 1;
  }

 if ((temp = fopen("temp.adm", "w+"))== NULL)
  {
    fprintf(stderr, "Imposible abrir archivo.\n");
    return 1;
  }

 fseek(hosts, 0, 0);
 fread(h_pol.host, sizeof(h_pol.host), 1, hosts);
 posicion=ftell(hosts);
 fseek(hosts, posicion, 0);
while( strcmp(argv[1],h_pol.host) && ! feof(hosts) )
 {
  posicion=ftell(hosts);
  fseek(hosts,posicion+6,0);
  fread(h_pol.host, sizeof(h_pol.host), 1, hosts);
 }
if( feof(hosts) &&  strcmp(h_pol.host,argv[1]) )
{
 printf("\n host %s no existe \n",argv[1]);
 fclose(hosts);
 fclose(temp);
 exit(1);
```

```
 }
if ( !strcmp(h_pol.host,argv[1]) )
 do
 {
   printf("\n Desea borrar %s ? ",argv[1]);
   scanf("%s",resp);
 }while(strcmp(resp,"s")&&strcmp(resp,"S")&&strcmp(resp,"n")&&strcmp(resp,"N"));

 if( !strcmp(resp,"n") || !strcmp(resp,"N"))
  {
  fclose(hosts);
  fclose(temp);
  exit(1);
  }

fseek(hosts,0,0);
 while( ! feof(hosts) )
 {
  posicion=ftell(hosts);
  fseek(hosts,posicion,0);
  fread(h_pol.host, sizeof(h_pol.host), 1, hosts);
  posicion=ftell(hosts);
  fseek(hosts,posicion,0);
  fread(h_pol.mls, sizeof(h_pol.mls), 1, hosts);
  posicion=ftell(hosts);
  fseek(hosts,posicion,0);
  fread(h_pol.com, sizeof(h_pol.com), 1, hosts);
  posicion=ftell(hosts);
  fseek(hosts,posicion,0);
  fread(h_pol.fin, sizeof(h_pol.fin), 1, hosts);
  if( strcmp(argv[1],h_pol.host) && strcmp(h_pol.host,host) )
   {
        fwrite(h_pol.host, sizeof(h_pol.host), 1, temp);
        fwrite(h_pol.mls, sizeof(h_pol.mls), 1, temp);
        fwrite(h_pol.com, sizeof(h_pol.com), 1, temp);
        fwrite(h_pol.fin, sizeof(h_pol.fin), 1, temp);
   }
 strcpy(host,h_pol.host);

}
  fclose(hosts);
  fclose(temp);

if ((hosts = fopen("politica.adm", "w+"))== NULL)
 {
   fprintf(stderr, "Imposible abrir archivo.\n");
   return 1;
 }

if ((temp = fopen("temp.adm", "r+"))== NULL)
 {
   fprintf(stderr, "Imposible abrir archivo.\n");
   return 1;
 }
strcpy(host,"");
strcpy(h_pol.host,"");
strcpy(h_pol.mls,"");
```

```
  strcpy(h_pol.com,"");
  strcpy(h_pol.fin,"");

  fseek(temp,0,0);
   while( ! feof(temp) )
   {
    posicion=ftell(temp);
    fseek(temp,posicion,0);
    fread(h_pol.host, sizeof(h_pol.host), 1, temp);
    posicion=ftell(temp);
    fseek(temp,posicion,0);
    fread(h_pol.mls, sizeof(h_pol.mls), 1, temp);
    posicion=ftell(temp);
    fseek(temp,posicion,0);
    fread(h_pol.com, sizeof(h_pol.com), 1, temp);
    posicion=ftell(temp);
    fseek(temp,posicion,0);
    fread(h_pol.fin, sizeof(h_pol.fin), 1, temp);
    if(strcmp(host,h_pol.host))
     {
          fwrite(h_pol.host, sizeof(h_pol.host), 1, hosts);
          fwrite(h_pol.mls, sizeof(h_pol.mls), 1, hosts);
          fwrite(h_pol.com, sizeof(h_pol.com), 1, hosts);
          fwrite(h_pol.fin, sizeof(h_pol.fin), 1, hosts);
     }
    strcpy(host,h_pol.host);
  }

    fclose(hosts);
    fclose(temp);
    return 0;
}
```

/* programa para ingresar puertos */

ptoins.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  FILE * ap;
  int posicion;
  char puerto[12];
  char host[12];

  if (argc!=3)
    {
        printf("\n numero incorrecto de parametros");
        exit(1);
    }
  if ((ap = fopen("puertos.adm", "a+"))== NULL)
  {
    fprintf(stderr, "Imposible abrir archivo.\n");
    return 1;
  }

  fseek(ap, 0, 0);
  fread(host, sizeof(host), 1, ap);
  fread(puerto, sizeof(puerto), 1, ap);
  posicion=ftell(ap);
  fseek(ap, posicion, 0);
 while( strcmp(argv[1],host) && strcmp(argv[2],puerto) &&! feof(ap) )
  {
   posicion=ftell(ap);
   fseek(ap,posicion,0);
   fread(host, sizeof(host), 1, ap);
   fread(puerto, sizeof(puerto), 1, ap);
  }

if ( !strcmp(host,argv[1]) )
  {
   printf("\n host %s ya existe \n",host);
   fclose(ap);
   exit(1);
  }
if ( !strcmp(puerto,argv[2]) )
  {
   printf("\n acceso negado! puerto %s asociado a otro host \n",argv[2]);
   fclose(ap);
   exit(1);
  }
 strcpy(host,argv[1]);
 strcpy(puerto,argv[2]);
 fwrite(host, sizeof(host), 1, ap);
 fwrite(puerto, sizeof(puerto), 1, ap);
 fclose(ap);
 return 0;
  }
```

/* programa para cambiar puerto asociado a un host */

ptocam.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  FILE * ap;
  int posicion, pos_cambio;
  char res;
  char puerto[12];
  char host[12];
  char nuevo_host[12];
  char nuevo_puerto[12];

  if (argc!=3)
    {
        printf("\n numero incorrecto de parametros \n");
        exit(1);
    }
  if ((ap = fopen("puertos.adm", "r+b"))== NULL)
  {
    fprintf(stderr, "Imposible abrir archivo.\n");
    return 1;
  }

 if( strcmp(argv[2],"h") && strcmp(argv[2],"p") )
  {
  printf("\n parametro no valido \n");
  exit(1);
  }
  fseek(ap, 0, 0);
  fread(host, sizeof(host), 1, ap);
 while( strcmp(argv[1],host) && ! feof(ap) )
  {
   posicion=ftell(ap);
   fseek(ap,posicion+12,0);
   fread(host, sizeof(host), 1, ap);
  }
if ( strcmp(host,argv[1]) )
  {
   printf("\n host %s no existe \n",argv[1]);
   fclose(ap);
   exit(1);
  }
 pos_cambio=ftell(ap);
if ( !strcmp(argv[2],"h") )
 {
 printf("\n nuevo host-> ");
 gets(nuevo_host);
 fseek(ap, 0, 0);
 fread(host, sizeof(host), 1, ap);
 while( strcmp(nuevo_host,host) && ! feof(ap) )
 {
  posicion=ftell(ap);
```

```
   fseek(ap,posicion+12,0);
   fread(host, sizeof(host), 1, ap);
  }
  if(!strcmp(host,nuevo_host) )
   {
    printf("\n host %s ya existe \n",nuevo_host);
    fclose(ap);
    exit(1);
   }
  fseek(ap,pos_cambio-12,0);
  fwrite(nuevo_host,sizeof(nuevo_host),1, ap);
 }

 if ( !strcmp(argv[2],"p") )
  {
  printf("\n nuevo puerto-> ");
  gets(nuevo_puerto);
  fseek(ap, 12, 0);
  fread(puerto, sizeof(puerto), 1, ap);
  while( strcmp(nuevo_puerto,puerto) && ! feof(ap) )
   {
   posicion=ftell(ap);
   fseek(ap,posicion+12,0);
   fread(puerto, sizeof(puerto), 1, ap);
   }
  if(!strcmp(puerto,nuevo_puerto))
   {
    printf("\n puerto %s ya existe \n",nuevo_puerto);
    fclose(ap);
    exit(1);
   }
  fseek(ap,pos_cambio,0);
  fwrite(nuevo_puerto,sizeof(nuevo_puerto),1, ap);
 }
 fclose(ap);
 return 0;
}
```

/* programa para eliminar puerto - host */

ptoeli.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  FILE *pto, *temp;
  char host[12];
  char host_ant[12];
  char puerto[12];
  int posicion;
  char resp[2];

 if (argc!= 2)
   {
    printf("\numero incorrecto de parametros ");
    exit(1);
   }
 if ((pto = fopen("puertos.adm", "r+"))== NULL)
  {
    fprintf(stderr, "Imposible abrir archivo.\n");
    return 1;
  }
 if ((temp = fopen("temp.adm", "w+"))== NULL)
  {
    fprintf(stderr, "Imposible abrir archivo.\n");
    return 1;
  }
 fseek(pto, 0, 0);
 fread(host, sizeof(host), 1, pto);
 posicion=ftell(pto);
 fseek(pto, posicion, 0);
 while( strcmp(argv[1],host) && ! feof(pto) )
  {
   posicion=ftell(pto);
   fseek(pto,posicion+12,0);
   fread(host, sizeof(host), 1, pto);
  }
 if( strcmp(host,argv[1]) )
 {
  printf("\n host %s no existe \n",argv[1]);
  fclose(pto);
  fclose(temp);
  exit(1);
 }
 if ( !strcmp(host,argv[1]) )
  do
  {
    printf("\n Desea borrar %s ? ",argv[1]);
    scanf("%s",resp);
  }while(strcmp(resp,"s")&&strcmp(resp,"S")&&strcmp(resp,"n")&&strcmp(resp,"N"));
  if( !strcmp(resp,"n") || !strcmp(resp,"N"))
   {
    fclose(pto);
```

```
    fclose(temp);
    exit(1);
  }
 strcpy(host_ant,"");
 fseek(pto,0,0);
  while( ! feof(pto) )
  {
   posicion=ftell(pto);
   fseek(pto,posicion,0);
   fread(host, sizeof(host), 1, pto);
   posicion=ftell(pto);
   fseek(pto,posicion,0);
   fread(puerto,sizeof(puerto),1,pto);
   if( strcmp(host,host_ant) && strcmp(host,argv[1]) )
    {
         fwrite(host, sizeof(host), 1, temp);
         fwrite(puerto, sizeof(puerto), 1, temp);
    }
   strcpy(host_ant,host);
 }
   fclose(pto);
   fclose(temp);
 if ((pto = fopen("puertos.adm", "w+"))== NULL)
  {
   fprintf(stderr, "Imposible abrir archivo.\n");
   return 1;
  }
 if ((temp = fopen("temp.adm", "r+"))== NULL)
  {
   fprintf(stderr, "Imposible abrir archivo.\n");
   return 1;
  }
 strcpy(host,"");
 strcpy(puerto,"");
 strcpy(host_ant,"");

 fseek(temp,0,0);
  while( ! feof(temp) )
  {
   posicion=ftell(temp);
   fseek(temp,posicion,0);
   fread(host, sizeof(host), 1, temp);
   posicion=ftell(temp);
   fseek(temp,posicion,0);
   fread(puerto, sizeof(puerto), 1, temp);

   if( strcmp(host,host_ant) )
    {
         fwrite(host, sizeof(host), 1, pto);
         fwrite(puerto, sizeof(puerto), 1, pto);
    }
   strcpy(host_ant,host);
 }
   fclose(pto);
   fclose(temp);
   return 0;
}
```

/* programa para listar usuarios del sistema */

usulist.c

```
#include <stdio.h>

int main()
{
  FILE *atrib;
  int posicion;
  char us_id[10];
  char us_id_ant[10];


  if ((atrib = fopen("atrib.adm", "r"))== NULL)
  {
    fprintf(stderr, "Imposible abrir archivo.\n");
    return 1;
  }

  fseek(atrib, 0, 0);
  fread(us_id, sizeof(us_id), 1, atrib);
  printf("\n %s",us_id);
  strcpy(us_id_ant,us_id);
 while( ! feof(atrib) )
  {
   posicion=ftell(atrib);
   fseek(atrib,posicion+86,0);
   fread(us_id, sizeof(us_id), 1, atrib);
   if (strcmp(us_id,us_id_ant) )
         printf("\n %s",us_id);
   strcpy(us_id_ant,us_id);
  }

 fclose(atrib);
 return 0;
}
```

/* programa para listar host del sistema */

hostlist.c

```
#include <stdio.h>

int main()
{
  FILE *hosts;
  int posicion;
  char host[12];
  char host_ant[12];

  if ((hosts = fopen("politica.adm", "r+"))== NULL)
  {
    fprintf(stderr, "Imposible abrir archivo.\n");
    return 1;
  }

  fseek(hosts, 0, 0);
  fread(host, sizeof(host), 1, hosts);
  printf("\n %s",host);
  strcpy(host_ant,host);
 while( ! feof(hosts) )
  {
   posicion=ftell(hosts);
   fseek(hosts,posicion+6,0);
   fread(host, sizeof(host), 1, hosts);
   if( strcmp(host,host_ant) )
     printf("\n %s",host);
   strcpy(host_ant,host);
  }
  fclose(hosts);
  return 0;
}
```

/* programa para listar puertos del sistema */

ptolist.c

```
#include <stdio.h>

int main()
{
  FILE *ap;
  char host[12];
  char host_ant[12];
  char puerto[12];
  char puerto_ant[12];
  int posicion,lon,pos,i;


  if ((ap = fopen("puertos.adm", "r+"))== NULL)
  {
    fprintf(stderr, "Imposible abrir archivo.\n");
    return 1;
  }
  fseek(ap, 0, 0);
  fread(host, sizeof(host), 1, ap);
  fread(puerto, sizeof(puerto),1,ap);
  if ( strcmp(host, host_ant) && strcmp(puerto,puerto_ant) )
   {
    printf("\n%s",host);
          lon=strlen(host);
          pos=15-lon;
          for(i=1;i<=pos;i++)
            printf(" ");

    printf("%s",puerto);
   }
  strcpy(host_ant,host);
 while( ! feof(ap) )
  {
   posicion=ftell(ap);
   fseek(ap,posicion,0);
   fread(host, sizeof(host), 1, ap);
   fread(puerto, sizeof(puerto), 1, ap);
   if ( strcmp(host, host_ant) && strcmp(puerto,puerto_ant) )
    {
          printf("\n%s",host);
          lon=strlen(host);
          pos=15-lon;
          for(i=1;i<=pos;i++)
            printf(" ");
          printf("%s",puerto);
    }
  strcpy(host_ant,host);
  }
 fclose(ap);
 return 0;
}
```

A 30