INTERFAZ *AIS* PARA APLICACIONES EN CAD/CAM/CG*

Oscar E. Ruiz S. Profesor Asistente
oruiz@sigma.eafit.edu.co
Jairo A. Saldarriaga. Asistente de investigación
jsaldarr@sigma.eafit.edu.co

Universidad EAFIT
Centro Interdisciplinario de Investigaciones
A.A. 3300 Medellín
COLOMBIA

RESUMEN

Los modeladores geométricos más comunes en el mercado ofrecen, además de sus servicios de modelado, una *API* (*Application Programming Interface*) que permite la construcción de aplicaciones o *software* cliente. Estas aplicaciones aprovechan los servicios básicos del modelador para proveer tareas específicas. Sinembargo las diferencias entre *API´s* de distintos modeladores imposibilita el intercambio del software cliente entre ellos.

Application Interface Specification -AIS- es una *API* genérica para ser usada por aplicaciones cliente de los modeladores geométricos. Este artículo reporta la implementación de *AIS* sobre *AutoCAD®* y *MicroStation®* y discute aspectos importantes de dicha implementación. Además presenta una aplicación cliente neutra que habla lenguaje *AIS* y por lo tanto se ejecuta transparentemente sobre los dos modeladores.

AIS se presenta como una alternativa económica para escribir aplicaciones de *CAD/CAM/CG*. Futuros desarrollos incluyen la implementación de *AIS* para labores gráficas y/o de base de datos.

I. INTRODUCCION

AIS es una descripción de la funcionalidad que un modelador geométrico 3D debe poseer para ser considerado como tal. Al ser una descripción funcional (prototipos de funciones) permite la implementación de una capa de aislamiento entre una aplicación cliente y el servidor específico de servicios de modelado 3D que atiende tal demanda.

Las presentes implementaciones de *AIS* sobre *AutoCAD®* y *MicroStation®* usan estos modeladores como servidores de funciones de modelado 3D. Con esto se logra la posibilidad de escribir aplicaciones cliente sobre la Interfaz Estándar en lugar de hacerlo directamente sobre el servidor que atiende la demanda. Una aplicación escrita en *AIS* se ejecuta independientemente de la interfaz funcional del modelador específico sobre el cual está corriendo, si se compromete a hacer requerimientos de servicios únicamente a través de la interfaz *AIS*.

II. REVISIÓN BIBLIOGRAFICA

AIS es un estándar diseñado por *CAM-I* (*Consortium for Advanced Manufacturing - International*) [6,7] como respuesta a la necesidad de implementar aplicaciones que puedan ser compartidas por usuarios que utilicen diferentes sistemas *CAD/CAM/CG* de modelado de sólidos. En esta investigación, dos modeladores fueron utilizados como servidores de servicios geométricos para implementar *AIS*: *AutoCAD®* [10] y *MicroStation®* [11]. Ambos modeladores utilizan ACIS® como *kernel* o motor geométrico [4,5], aunque sus respectivas *API´*s difieren notablemente [8,9].

La implementación de funciones para manipulación de entidades geométricas está basada en la fundamentación teórica de computación gráfica [1], geometría computacional [3] y modelado de sólidos [2]. La Figura 1 muestra la funcionalidad de *AIS* implementado sobre dos modeladores genéricos utilizados como servidores de funciones de modelado 3D.

III. AIS

AIS constituye una interfaz *funcional* que define el comportamiento de las entidades en un modelador de sólidos. No es una interfaz para definir *datos* detallados de cada tipo de entidad. Sólo datos neutros (*handles*) pasan a través de la interfaz. La versión de *AIS* implementada (2.1) trabaja según los modelos de representación *B-rep* (*manifold boundary representation*) y *CSG (constructive solid geometry*) aceptados por la academia y la industria [2].

* Para publicación en EGRAF-97: I Simposio Iberoamericano de Expresión Gráfica en las Ingenierías y la Arquitectura. Camaguey, Cuba, Cuba. Oct 8-15, 1997

Los tipos de entidades y funciones de *AIS* están diseñados para ser soportados por todos los modeladores que siguen las representaciones indicadas (*B-rep y CSG*).

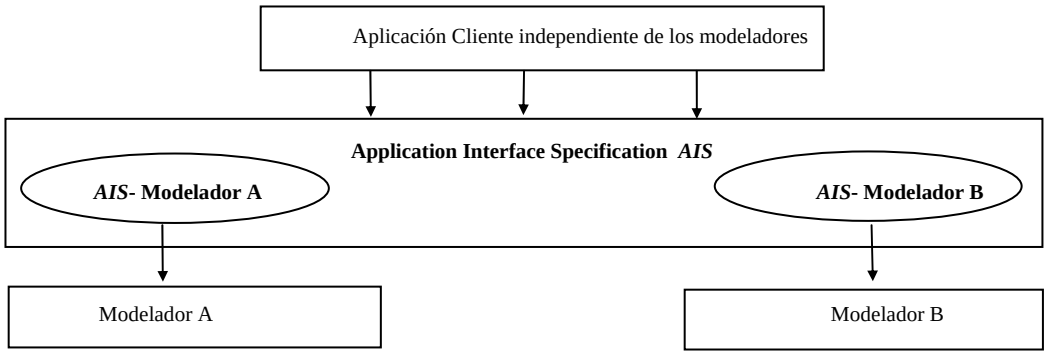


Figura 1. *AIS* implementado sobre dos servidores CAD

AIS no especifica funciones de graficado, control o interacción con el usuario. Sólo especifica las funciones necesarias para construir y consultar la información geométrica y topológica. Sin embargo, *AIS* contempla el almacenamiento de atributos que pueden ser usados posteriormente para control y graficado. Algunos atributos implementados almacenan Nombre, Color y Visibilidad de las entidades geométricas.

Los tipos de entidades *AIS* son definidos principalmente por sus relaciones y funciones, por lo cual es posible que no correspondan exactamente a las entidades del modelador. Ciertas entidades *AIS* pueden tener equivalentes múltiples en el modelador y viceversa. Ej: Las entidades *AIS* `Cartesian_Point`, `Point_on_Curve` y `Point_on_Surface` pueden estar representadas por la misma entidad del modelador `MODELER_POINT`. La especificación de la correspondencia entre estas entidades es inherente a cada implementación *AIS* y debe ser transparente a cualquier aplicación cliente de *AIS* .

Debido a la diferencia en las capacidades de los modeladores y los requerimientos de las aplicaciones, *AIS* define 31 *categorías* [6] para las funciones específicas que soporta un modelador determinado. La categoría 0 especifica los servicios mínimos de un modelador; categorías crecientes contemplan servicios cada vez más sofisticados. En esta investigación se implementaron funciones de 15 categorías que abarcan creación y manipulación de entidades geométricas y topológicas.

Como especificación funcional, *AIS* describe sus funciones independientemente de lenguajes de programación específicos [6]. Sin embargo, la implementación de *AIS* en lenguaje C es sugerida por *CAM-I*.

AIS está organizado con una filosofía orientada a objetos, en la cual las funciones se agrupan bajo sus tipos de entidad primarios. Los tipos de entidades están organizados en una jerarquía de clases, como puede verse en la Figura 2.

En esta estructura jerárquica, las entidades están agrupadas bajo los conceptos de *encapsulación* y *herencia*. Estas entidades reflejan los conceptos de *topología* y *geometría* usados en el modelado de sólidos [5]. Esta organización jerárquica y modular de las entidades *AIS* hace que las entidades del modelador sean "actores", que asumen el papel de varias entidades *AIS* .

1) *Encapsulación y herencia. AIS* esta construido de (a) tipos y subtipos de entidades y (b) funciones. Algunos tipos no son entidades propiamente dichas, sino simplemente *generalizaciones* de subtipos que identifican las funciones que comparten todos sus subtipos. Las entidades se crean de subtipos específicos y no de tipos generalizados. Hay dos tipos de herencia: (i) primaria y (ii) secundaria.

i) La *herencia primaria* especifica que una entidad depende jerárquicamente de otra, y que todas las funciones de esa instancia superior son aplicables a ella. Se dice entonces que la primera entidad *hereda* de la segunda. Ej: *Línea* "hereda" las funciones de *Curva* (Líneas continuas en la Figura 2)

ii) La *herencia secundaria* permite a una entidad la utilización de las funciones de otra entidad, aunque no haya una relación jerárquica entre ellas. Ej: Las funciones aplicables a una superficie esférica son aplicables a una esfera; pero ésta no es subordinada a aquella. (Líneas punteadas en la Figura 2)

2) *Topología y Geometría*. En un modelo *B-rep* hay dos tipos definidos de entidades: (i) *Topologías* y (ii) *Geometrías*.

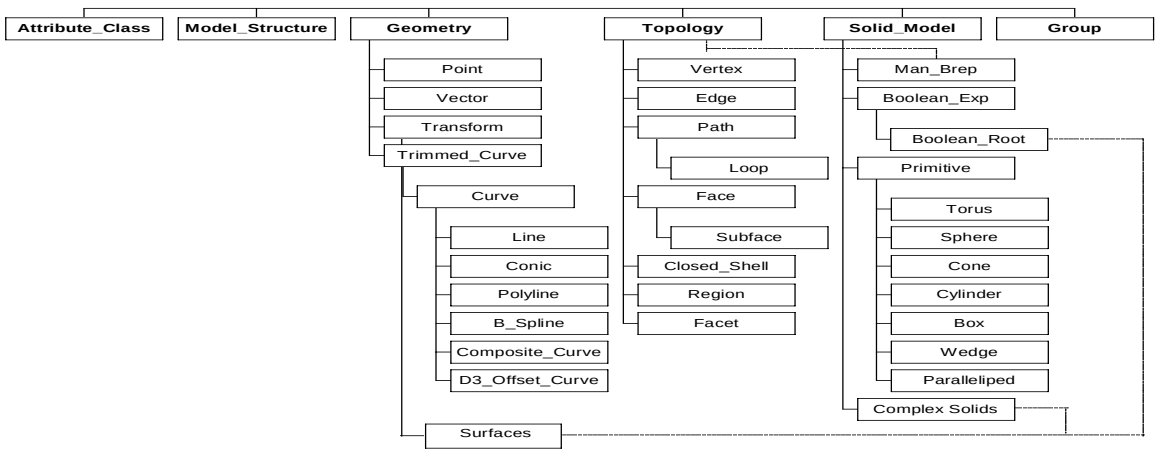


Figura 2. Estructura jerárquica de clases *AIS*

i) La *topología* de un objeto especifica las relaciones de conexión y frontera entre sus partes. *AIS* define otros tipos de entidades topológicas más complejas como *Closed_Shell*, *Region* y *Facet*. Las relaciones entre los tipos de entidades topológicas *AIS* pueden apreciarse en la figura 2.

ii) La *geometría* define matemáticamente a las entidades topológicas. Cada topología tiene una geometría portadora. Así, la geometría asociada a una *Cara* es una *Superficie*, una *Arista* se asocia con una *Curva* y un *Vértice* con un *Punto*. Por ejemplo, en una pirámide las caras tienen asociada la geometría plano, las aristas la geometría línea y los vértices la geometría punto. En este caso, una cara de la pirámide está delimitada por un anillo (*loop)*, que la limita y la conecta con las otras 3 caras. Cada *loop* se compone de 3 aristas (*edges*) que poseen a su vez vértices (*vertex*) en sus extremos.

AIS define un grupo de funciones genéricas y tipos de datos provistos en la especificación funcional. Dicho grupo se encuentra en librerías C y es una *API* fuente estandarizada para los modeladores y debe ser implementada por el desarrollador de la Intefaz *AIS* para un modelador específico.

La portabilidad de las aplicaciones desarrolladas en *AIS*-C se da a nivel de código fuente (no binario). Una aplicación *AIS*-C desarrollada para un modelador específico puede recompilarse y encadenarse en otra implementación *AIS*-C de un modelador diferente. Si ambas implementaciones *AIS*-C soportan la mismas categorías y tipos de entidades *AIS*, la aplicación se ejecuta correctamente.

IV. *AIS* sobre AutoCAD

La implementación de *AIS* sobre AutoCAD se hizo a través de su *API ARX* (*AutoCAD Run Extension Programming Environment*). Esta *API*, organizada en una arquitectura orientada a objetos, utiliza el lenguaje de programación C++. Nótese que aunque los prototipos de funciones *AIS* son en C, su implementación interna puede ser hecha en C++. Las librerías *ARX* incluyen un conjunto de herramientas que aprovechan la arquitectura abierta de *AutoCAD* para proveer acceso directo a sus estructuras de bases de datos, interfaz gráfica y definición de comandos nativos.

Una aplicación *ARX* es una *DLL* (*Dynamic Link Library*) que comparte espacio en memoria con *AutoCAD* y hace llamadas directas a sus funciones. Las librerías de *ARX* incluyen macros para facilitar la definición de nuevas clases; además permiten aumentar funcionalidad a las clases existentes.

La estructura jerárquica de clases *ARX* está compuesta por dos grandes grupos independientes: Clases *AcRx* y Clases *AcGe* (Ver figura 3). El grupo principal *AcRx* está compuesto por todas las clases necesarias para realizar aplicaciones. El grupo de clases *AcGe* permite la creación y manipulación de instancias geométricas de los objetos. Algunos de los objetos *AcRx* pueden ser construidos a partir de objetos *AcGe.*

AcRx tiene una superclase (`AcRxObject`) que abarca todas las demás. Ella suministra una clase básica para el manejo de la base de datos: (i) Creación y manipulación de Objetos

(`AcDbObject`) y (ii) Creación y manipulación de Entidades (`AcDbEntity`). Las clases restantes

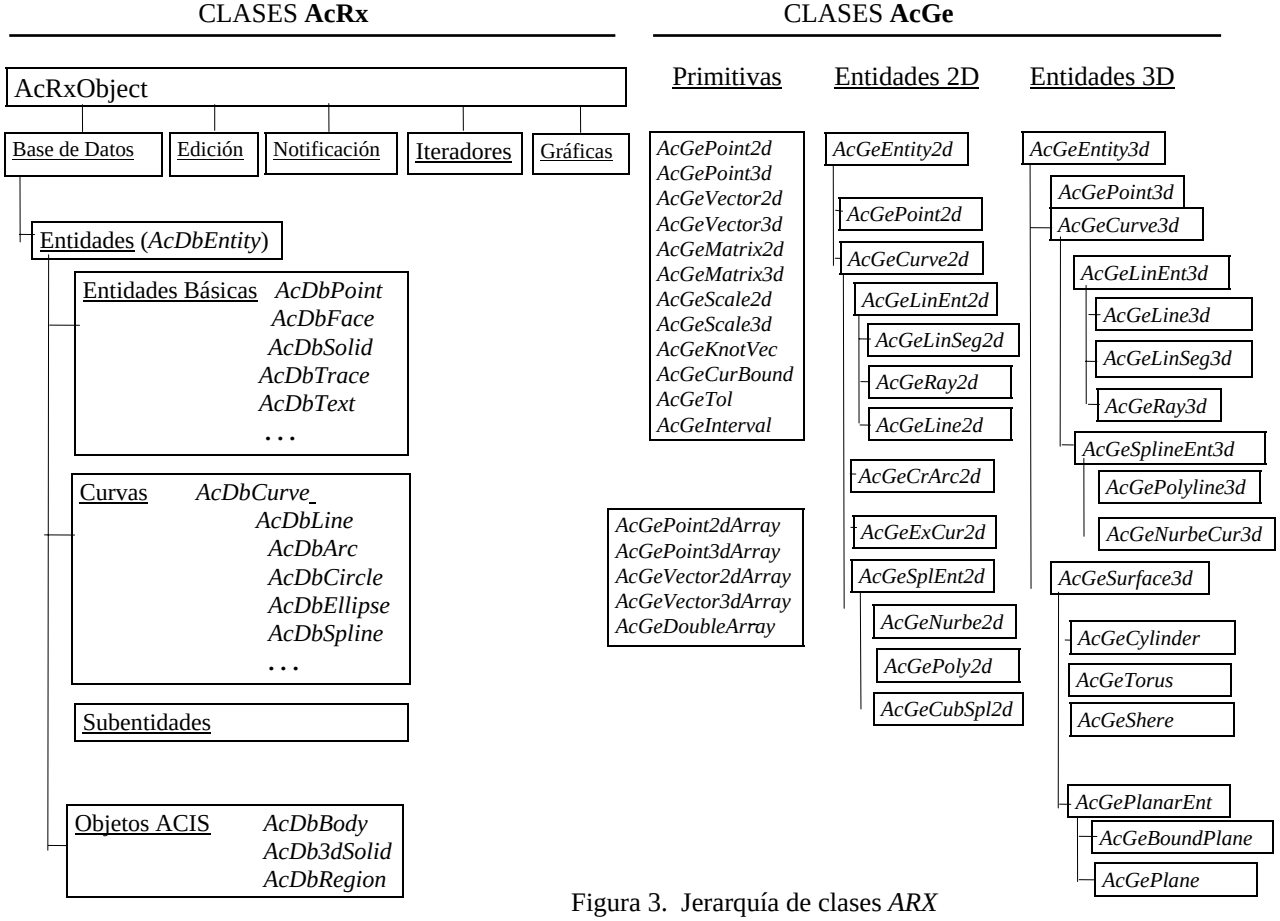


Figura 3. Jerarquía de clases *ARX*

permiten el manejo de edición, notificación de eventos, interfaz gráfica y control en tiempo de ejecución, entre otros.

Las clases *AcGe* están agrupadas en tres bloques: El primero permite la creación de entidades primitivas genéricas como Puntos, Vectores y Matrices; los otros dos contienen clases para construcción y manipulación de entidades 2D y 3D respectivamente.

En los diagramas mostrados en la figura 3 las líneas indican la relación "es un". Por ejemplo un *AcDbPoint* es un *AcDbEntity* el cual es un *AcDbObject* que a su vez es un *AcRxObject*.

La implementación de *AIS* para un modelador
requiere que las entidades definidas por la especificación estén soportadas por objetos en dicho modelador. No es posible crear un objeto *AIS* si no existe una entidad (o entidades) con el mismo comportamiento en el modelador.

La siguiente comparación entre entidades *AIS* y entidades *ARX* se hace por clases. El punto de partida son las clases *AIS* (ver Figura 2).

Para las principales clases *AIS* se describen las clases *ARX* que la soportan.

1.`GEOMETRY`. Para la creación de entidades primitivas *AIS* se utiliza básicamente la librería **`AcDb`**, ya que sus objetos tienen una representación gráfica administrada por el modelador. Algunos objetos **`AcDb`** se crean a partir de objetos **`AcGe`**. Las entidades *AIS* que no tienen soporte en la librería **`AcDb`** se crean directamente con objetos **`AcGe`**. Estas entidades no tienen representación gráfica. En general son objetos de geometría infinita. Ejemplos de ello son *Surface* y sus objetos derivados (*Plane*, *cylinder_Surf*, etc.)

2. `TOPOLOGY`. *ARX* no permite la *creación* de las entidades topológicas básicas. Por lo tanto, sólo se realizó la implementación de funciones de *consulta*. Estas fueron soportadas por objetos *ARX* de la librería **`AcBr`**, cuyos objetos no tienen

representación gráfica. 3. SOLID_MODEL. Esta clase *AIS* representa tipos de sólidos que participan en una estructura *CSG*

Entidad *AIS*	Entidades *ARX*	Comentarios
GEOMETRY		
Point	AcDbPoint	Derivada de AcDbEntity.
Cartesian Point	AcGePoint2d, AcGePoint3d	Entidades Primitivas AcGe
	AcGePointEnt2d, AcGePosition2d	Derivadas de AcGeEntity2d
	AcGePointEnt3d, AcGePosition3d	Derivadas de AcGeEntity3d
Point_on_Curve	AcGePointOnCurve2d AcGePointOnCurve3d	
Point_on_Surface	AcGePointOnSurface	
Projection_Point	No existe como entidad en *ARX*	Pueden encontrarse usando métodos de curvas y superf.
Intersection_Point	No existe como entidad en *ARX*	
Vector	AcGeVector2d, AcGeVector3d	Entidades *primitivas.*
Transform	AcGeMatrix2d, AcGeScale2d AcGeMatrix2d, AcGeScale3d	Entidades *primitivas.*
Trimmed Curve	---	
Curve	AcDbCurve	Entidad básica AcDb.
	AcGeCurve2d	Derivada de AcGeEntity2d
	AcGeCurve3d	Derivada de AcGeEntity3d
Line	AcDbLine,AcDbRay, AcGeLinearEnt2d, AcGeLine2d, AcGeLineSeg2d, AcGeRay2d, / AcDbXline, AcGeLinearEnt3d, AcGeLine3d, AcGeLineSeg3d, AcGeRay3d	Derivadas AcDbCurve.

Tabla 1. Correspondencia entre algunas entidades *ARX* y *AIS*

(provienen de operaciones booleanas). *ARX* no posee una clase equivalente; sin embargo muchos de los objetos de la clase *AIS* son soportados por la librería AcDb. Los objetos suministrados por *ARX* son basados directamente en *ACIS* [4,5] y derivados de AcDbEntity. Estos son AcDbBody, AcDb3dSolid y AcDbRegion. En la Tabla 1 puede verse un extracto de la correspondencia de entidades *AIS* y *ARX* [10]

V. *AIS* sobre MicroStation

MDL (*MicroStation Development Language)* es la *API* de *MicroStation*®. *MDL* permite a las aplicaciones aprovechar las capacidades de *ACIS* y suministra funciones adicionales más complejas que son provistas como ayuda para el desarrollador.

Además de las funciones de manejo de entidades, *MDL* tiene funciones matemáticas, de interacción con el usuario, gráficas, de texto, etc. Todas ellas pueden ser usadas en una misma aplicación, por lo cual el alcance de *MDL* va más allá de lo que ofrece el *kernel* geométrico en sí.

Las aplicaciones *MDL* están completamente integradas al entorno de *MicroStation*® y son independientes de las plataformas de hardware o software. Varios de los comandos y utilidades de *MicroStation*® son aplicaciones *MDL*.

MDL incluye una implementación interna de un compilador C y sus librerías. El compilador de *MDL* convierte los archivos fuente en *pseudocódigos* que *MicroStation*® comprende. La implementación C contiene: Un interpretador de pseudocódigos que ejecuta las aplicaciones *MDL*, compilador y encadenador de C, compilador de recursos, depurador, constructor de aplicaciones (*makefile interpreter*) y funciones predefinidas adicionales. Algunas características importantes de *MDL* son:

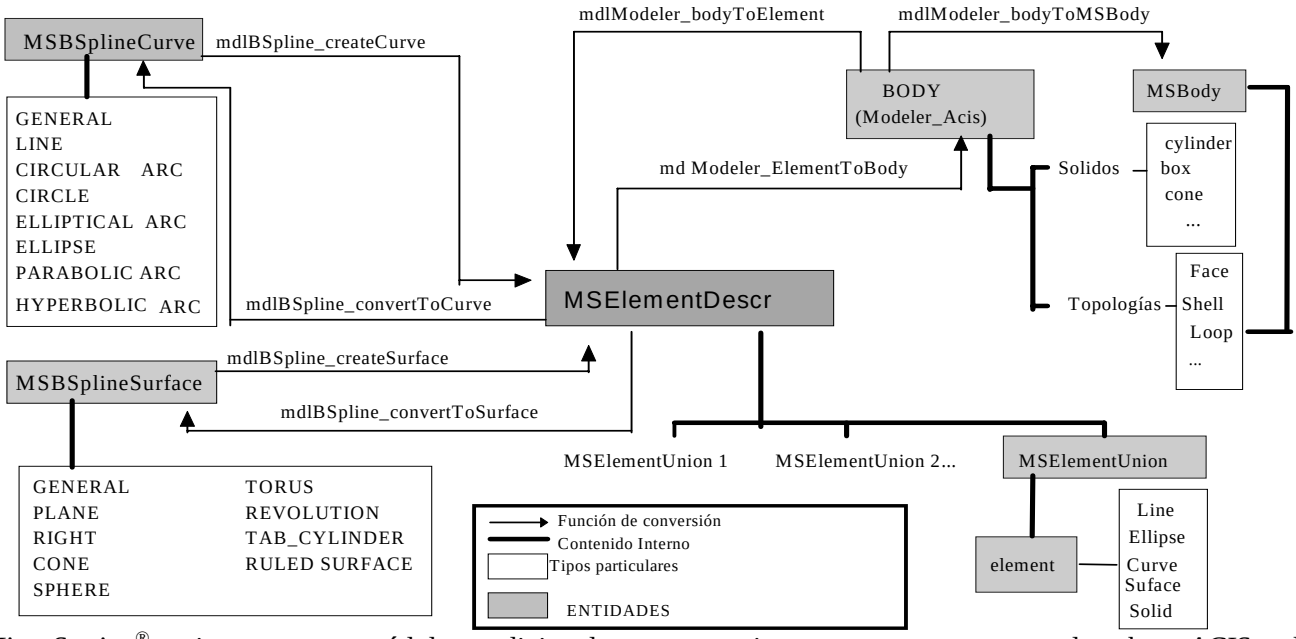
(i) Los *hooks* (apuntadores a funciones) pueden modificar el comportamiento de *MicroStation*. A través de los *hooks* una aplicación *MDL* puede definir que una función se ejecute cuando ocurra un evento específico (*event-driven programming*).

(ii) *MDL* permite desarrollar una interfaz gráfica del usuario (GUI) para las aplicaciones.

(iii) El *manejador de recursos* de *MDL* permite controlar datos que no están realmente incluidos en el programa C.

(iv) Pueden crearse librerías objeto (*object library*) en código binario para ser enlazadas en una o varias aplicaciones *MDL*, pero deben ser creadas con el

mismo compilador y plataforma de la aplicación.



MicroStation® tiene un módulo adicional que permite un mayor control sobre *ACIS*: el

Figura 4. Conversión y jerarquía entre tipos de Entidades *MicroStation*

MicroStation Modeler®. Para usuarios o aplicaciones especializadas (como el proyecto de implementación *AIS*) el *Modeler* es una herramienta necesaria.

Las ventajas del *Modeler* son (i) la representación topológica de los cuerpos de *MicroStation*® y (ii) la construcción de sólidos primitivos.

MDL no ofrece una relación jerárquica entre las entidades geométricas, pues no es un lenguaje orientado a objetos; todas ellas se agrupan a un mismo nivel. Por esto la clasificación de *geometrías* es muy general.

La implementación de las *topologías* se hace exclusivamente a través del *Modeler* y es más rigurosa en cuanto a la jerarquía de sus entidades.

En *MDL* las *geometrías* se clasifican en cinco tipos de entidades:

1) **`element`**: Entidades geométricas básicas (línea, círculo, superficies, etc.)

2) **`MSElementUnion`**: Es un *union* de C con todos los `element` posibles.

3) **`MSBspline`** (`Curve` y `Surface`) Entidades geométricas B-spline.

4) Sólidos primitivos del *Modeler* (`BODY`): Cilindros, cajas, pirámides, etc.

5) **`MSElementDesc`**: La más importante. No es propiamente una entidad geométrica; es un "*recipiente comodín*" que mediante las funciones apropiadas puede almacenar cualquiera de los cuatro tipos anteriores de entidades.

Una relación entre todos los tipos de entidades de *MicroStation*® y las formas de conversión entre ellas se muestra en la figura 4.

Para analizar la factibilidad de implementar *AIS* en *MDL* se tiene en cuenta: (i) las entidades que define *AIS* y las ofrecidas por *MDL;* (ii) el comportamiento funcional de estas entidades (*categorías AIS*); y (iii) aspectos técnicos del lenguaje de la *API* como tipos de datos, sintaxis y restricciones. En la Tabla 2 puede verse un extracto de la correspondencia de entidades *AIS* y *MDL* [11].

VI. EJEMPLO DE APLICACION *AIS* COMPATIBLE

El código mostrado a continuación implementa una aplicación simple escrita en *AIS*. Ella ejecuta correctamente sobre los dos modeladores para los cuales *AIS* está implementado. Aplicaciones mas serias han sido desarrolladas por esta investigación. Se destaca *HormaCAD*, una aplicación *AIS* compatible que construye superficies a partir de archivos de digitalización [10,11].

En el ejemplo se crean entidades simples *AIS* como líneas y círculos. Además se asignan a las entidades atributos de color y visibilidad. Durante la ejecución se activan y desactivan consecutivamente los atributos de las diferentes entidades.

Entidad *AIS*	Entidades *MDL*	Elegida	Comentarios
GEOMETRY ***Point***	---		MicroStation no contempla la entidad
Cartesian Point	Line, Point String.	**Line**	Point. Se puede lograr comportamiento
Point_on_Curve	Line, Point String.		de un punto con una línea de inicio y
Point_on_Surface	Line, Point String.		final iguales, o con una *cadena de*
Projection_Point	Line, Point String.		*puntos* de un solo elemento. Da mayor
Intersection_Point	Line, Point String.		funcionalidad la línea.
Vector	Dpoint3d		No es una entidad *permanente.*
Transform	Transform		No es una entidad *permanente.*
Trimmed Curve	---		
Curve	Curve,Bspline Curve		Bspline curve aproxima a las demás
Line	Line	**Line**	Segmento, no línea infinita. Pero sirve.

Tabla 2. Correspondencia entre algunas entidades *MDL* y *AIS*

Nótese que *AIS* sólo pasa *handles*. El contenido de los *handles* es privado para las funciones que implementa *AIS* sobre cada modelador. Por ejemplo la entidad *cara* declarada como *cit_hndl* (*handle AIS*) en la línea 6, se utiliza en la función *cici_create* (círculo *AIS*) de la línea 27. Esta función *AIS* construye un círculo en el modelador subyacente y retorna un apuntador en el *handle*. Adicionalmente se utilizan funciones *eafit_xxx*. Estas funciones de depuración gráfica se han definido con base en los *handles AIS*. Esto implica que también son portables. La Figura 5 muestra una simulación de la ejecución de la aplicación.

```
1  /*Función que opera con atributos. Realiza una animación sencilla que
     dibuja alternativamente caras serias y risueñas de diferentes colores.
     Las funciones cixx_xxx son implementaciones AIS sobre los
                                                        modeladores.*/
2    void risas()
3    {
4              int rc;
5              char kw[20];
6              cit_hndl cara, ojo_izq, ojo_der, boca_a, boca_c;
7              double radio_cara, radio_ojos, radio_boca_a;
8              cit_vect c_cara, c_ojo_izq, c_ojo_der, c_boca_a;
9              cit_vect boca_c1, boca_c2;
10             long conta = 0;
11             unsigned char risa = 0;
12             cit_attr atributos;
13             cit_vect axis_vect;
14             axis_vect[X] = 0.0; axis_vect[Y] = 0.0; axis_vect[Z] = 1.0;
15             radio_cara = 4.4;
16             c_cara[X] = 11; c_cara[Y] = 6; c_cara[Z] = 0;
17             radio_ojos = 0.68;
18             c_ojo_izq[X] = 9.6; c_ojo_izq[Y] = 7.6; c_ojo_izq[Z] = 0.0;
19             c_ojo_der[X] = 13.2; c_ojo_der[Y] = 7.6; c_ojo_der[Z] = 0.0;
20             radio_boca_a = 1.25;
21             c_boca_a[X] = 11.35; c_boca_a[Y] = 4.01; c_boca_a[Z] = 0.0;
22             boca_c1[X] = 9.6; boca_c1[Y] = 4.04; boca_c1[Z] = 0.0;
23             boca_c2[X] = 13.2; boca_c2[Y] = 4.04; boca_c2[Z] = 0.0;
24             /*Inicializar atributos*/
25             init_generic_attribs();
26             /* Crear entidades */
27             cici_create(radio_cara,c_cara, axis_vect, 0, NULL,cara);
28             eafit_AIS_display_entity(cara);
29             cici_create(radio_ojos, c_ojo_izq, axis_vect, 0, NULL,ojo_izq);
30             eafit_AIS_display_entity(ojo_izq);
31             cici_create(radio_ojos, c_ojo_der, axis_vect, 0, NULL, ojo_der);
32             eafit_AIS_display_entity(ojo_der);
33             cici_create(radio_boca_a, c_boca_a, axis_vect, 0, NULL, boca_a);
34             /* Cambiar atributos interactuando con usuario */
35             atributos.l = 0;
36             cien_set_attr(boca_a, visibility_attr_hndl, 1, atributos, 0);
37             cili_cre_pt_pt(boca_c1, boca_c2, boca_c);
38             eafit_AIS_display_entity(boca_c);
39             eafit_initget(0, "Parar Continuar");
40             c = eafit_getkword("Parar/<Continuar>: ", kw);
41            while (rc is RTNONE)
42            {
43                if (conta is 256)
44                {
45                  conta = 0;
46                }
47                 atributos.i = conta;
48                 cien_set_attr(cara, color_attr_hndl, 1, atributos, 0);
49                 eafit_AIS_display_entity(cara);
50                 if (not(risa))
51                 {
52                    risa = 1;
53                    atributos.l = 0;
54                    cien_set_attr(boca_c,visibility_attr_hndl, 1, atributos, 0);
55                    atributos.l = 1;
56                    cien_set_attr(boca_a, visibility_attr_hndl, 1, atributos,0);
57                    eafit_AIS_display_entity(boca_a);
58                 }
59                 else
60                 {
61                     risa = 0;
62                     atributos.l = 0;
63                     cien_set_attr(boca_a, visibility_attr_hndl, 1, atributos, 0);
64                     atributos.l = 1;
65                     cien_set_attr(boca_c, visibility_attr_hndl, 1, atributos, 0);
66                     eafit_AIS_display_entity(boca_c);
67                 }
68                conta++;
69                eafit_initget(0, "Parar Continuar");
70                rc = eafit_getkword("Parar/<Continuar>: ", kw);
71            }/*while*
72     }/*end of function*/
```

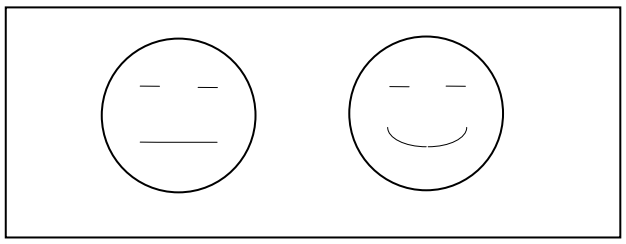


Figura 5. Simulación de la ejecución de la aplicación risas (*AIS* compatible)

VII. CONCLUSIONES

AIS proporciona una especificación funcional genérica de los servicios y entidades de un modelador geométrico. Puede considerarse como la especificación de una *API* estándar que permite al desarrollador concentrarse en los aspectos estrictamente geométricos de su aplicación, independientemente de los aspectos técnicos y características específicas de un sistema *CAD* en particular. La implementación de *AIS* en varios modeladores geométricos de difusión comercial en el medio latinoamericano constituye un cimiento sólido para el desarrollo futuro de aplicaciones en sistemas *CAD/CAM/CG* y para la generación de tecnología propia a partir de la tecnología existente.

El resultado de la implementación de *AIS* sobre *AutoCAD y MicroStation* es altamente satisfactorio. Se pueden desarrollar aplicaciones *AIS* que corran en los dos modeladores con absoluta transparencia y aplicables a las necesidades de la industria local.

i) *AIS* es una especificación apropiada para establecer una *API* estándar entre diferentes modeladores. Distingue claramente entre los servicios geométricos generales de cualquier sistema *CAD* y las características específicas de un sistema particular.

ii) La definición de entidades por comportamiento es un aspecto importante de la especificación. Permite usar entidades específicas de un modelador sin necesidad de que haya una correspondencia exacta con las entidades de otro; sólo se requiere que se *comporten* igual frente a las funciones del estándar.

iii) Una mejora importante sería la definición de funciones de interacción con el usuario. Se podría definir una especificación paralela o extender la original para que admita funciones de entrada y salida de datos. En el caso de extender la especificación actual, se podría adicionar *categorías AIS* que definieran el alcance de cada sistema *CAD*.

iv) La implementación del estándar *AIS* sobre los dos modeladores debe continuarse en el futuro. En este sentido tienen prioridad a) el desarrollo de una base de datos que permita almacenar y recuperar las entidades *AIS* y b) las operaciones booleanas entre sólidos.

v) Nótese que *AIS* basado en C fue exitosamente implementado junto con aplicaciones cliente sobre *ARX* (basado en C++) y con *MDL* (basado en C). Por otra parte fue compilado con MSVC++® (*ARX*) y con el compilador interno de C para *MDL*. Ello muestra que es un estándar verdaderamente portable.

VIII. REFERENCIAS BIBLIOGRAFICAS

1. Foley-VanDame-Feiner-Hughes. *COMPUTER GRAPHICS: Principles and Practice*. 2da. Edición. Addison Wesley. USA, 1991
2. Marti Mantyla. *An Introduction to SOLID MODELING*. Series Editors. USA, 1988
3. Michael E. Mortenson. *GEOMETRIC MODELING*. John Wiley. USA, 1985
4. Spatial Technology Inc. *ACIS, The Standard*. Applied Geometry Corp. USA, 1995
5. Spatial Technology Inc. *GEOMETRIC MODELER. Technical overview* Applied Geometry Corp. USA, 1995
6. Paul Ranyak. CAM-I (Consortium for Advanced Manufacturing - International) *Application Interface Specification (AIS). Versión 2.1 Vol. I (Functional Specification).* Integrity Systems. USA, 1994
7. Paul Ranyak. CAM-I (Consortium for Advanced Manufacturing - International) *Application Interface Specification (AIS). Versión 2.1 Vol. II (C Language Binding).* Integrity Systems. USA, 1994
8. Autodesk, Inc. *AutoCAD Release13 ARX Developer's Guide*. USA, 1996
9. Bentley Systems, Inc. *MicroStation Development and Support Guide*. USA, 1995
10. Darío Isaza, Jairo Saldarriaga. *Interfaz AIS para AutoCAD R13*. Proyecto de grado. Universidad EAFIT. Medellín, 1997
11. Jorge León Posada. *Intefaz AIS para MicroStation V5*. Proyecto de grado. Universidad EAFIT. Medellín, 1997