MÉTODOS NUMÉRICOS EN INGENIERÍA V
J. M. Goicolea, C. Mota Soares, M. Pastor y G. Bugeda (Eds.)

MECHML: UN NUEVO LENGUAJE BASADO EN XML PARA LA DESCRIPCIÓN DE SISTEMAS MECÁNICOS MULTI-CUERPO

M. González*, E. Álvarez y J. García de Jalón****

* Laboratorio de Ingeniería Mecánica - Departamento de Ingeniería Industrial II
Universidad de La Coruña
Mendizábal s/n, 15403 Ferrol, España
e-mail: lolo@cdf.udc.es, web http://lim.ii.udc.es

** Escuela Técnica Superior de Ingenieros Industriales
Universidad Politécnica de Madrid
José Gutiérrez Abascal 2, 28006 Madrid, España
e-mail: jgjalon@etsiit.upm.es, web http://mat21.etsii.upm.es/mbs

Palabras clave: MechML, Sistemas Multicuerpo, Lenguaje de Modelado, XML.

Resumen. *Actualmente no existe un lenguaje estándar para modelar sistemas mecánicos multicuerpo, y la diversidad de formatos existentes dificulta enormemente la colaboración entre grupos de trabajo. Este artículo presenta MechML (Mechanism Markup Language), un lenguaje de modelado de mecanismos basado en XML que presenta ventajas significativas sobre otros formatos. Aunque MechML todavía es un prototipo que necesita ser mejorado y ampliado para tener aplicación industrial, sus posibilidades futuras resultan muy atractivas. Una de las aplicaciones más interesantes es establecer benchmarks y colecciones de problemas difíciles que constituyen un reto para todos los que investigan en esa área.*

1. INTRODUCCIÓN

En los últimos años el creciente desarrollo de las tecnologías de la información ha propiciado la aparición de nuevas formas de trabajo en las tareas de diseño y fabricación de sistemas mecánicos. Estas nuevas tecnologías permiten a grupos geográficamente dispersos cooperar entre sí estrechamente y llevar a cabo proyectos industriales y de investigación de mayor envergadura. En este contexto de ingeniería concurrente el intercambio de información técnica de manera sencilla y eficiente es de capital importancia, y desde hace varios años se vienen desarrollando estándares para modelar información relativa al CAD y al FEM.

Sin embargo, en el campo de la simulación de sistemas multicuerpo no existe un formato de datos estándar para la definición de mecanismos: las aplicaciones comerciales y los grupos de investigación que desarrollan software de simulación utilizan formatos propios e incompatibles entre sí. Esta diversidad de formatos trae desventajas importantes: se multiplica el número de traductores necesarios, dificulta la reutilización de modelos, y, en general, dificulta la cooperación entre distintos grupos de trabajo. Esta situación hace evidente la necesidad de disponer de un formato estándar para la definición de sistemas mecánicos multicuerpo.

A lo largo de la última década ha habido diversos esfuerzos para definir un lenguaje de modelado de mecanismos universal: STEP Part 105 [1], Dymola [2], Modelica [3], etc. A pesar de ello ninguna de las propuestas ha cuajado en el ámbito académico o industrial debido a la falta de software que facilite el tratamiento de la información expresada en estos formatos.

Este artículo presenta MechML (Mechanism Markup Language), un prototipo de lenguaje de modelado de sistemas multicuerpo basado en XML que aporta ventajas significativas sobre otros formatos existentes. Su sencillez, extensibilidad y facilidad de uso hacen de él un buen candidato para convertirse en el formato estándar en la industria de simulación de mecanismos.

2. DESCRIPCIÓN GENERAL

MechML está basado en XML (eXtensible Markup Language [4]), el formato universal para almacenar e intercambiar datos en la World Wide Web. La versión actual de XML fue aprobada en 1999 por el World Wide Web Consortium, integrado por más de 200 organizaciones relacionadas con Internet, con el objetivo de proporcionar a las empresas del sector un lenguaje de modelado sencillo, versátil y extensible.

XML no es un lenguaje en sí, sino un *metalenguaje*, es decir, un lenguaje para definir otros lenguajes con finalidades más específicas. Sus ventajas fundamentales son su sencillez, su gran integración con diversos estándares existentes (caracteres Unicode, referencias URL, números en coma flotante IEEE, fechas ISO, etc.) y el amplio soporte disponible (documentación, recursos on-line, software de edición y procesado, etc.). XML ha sido utilizado para definir lenguajes estándar en diversas áreas de conocimiento, tales como Matemáticas (MathML [5]), Química (CML [6]), Geografía (GML [7]) e Instrumentación (IML [8]). A continuación se describirá brevemente la tecnología XML,

mostrando con ejemplos las ventajas que ofrece.

2.1. Modelo de datos y sintaxis de XML

La estructura y sintaxis de un lenguaje basado en XML es sumamente sencilla y resulta familiar a cualquier persona que haya utilizado HTML. El siguiente fragmento de MechML muestra como se describiría la información inercial de un sólido rígido:

```
<body name="aBody">
  <inertia
    mass="3.2537E-5"
    cogx="0.05185"
    cogy="5.27520"
    cogz="0.0"
    Ixx="2.993963E-9"
    Iyy="5.885866E-8"
    Izz="6.131033E-8"/>
</body>
```

Todos los lenguajes basados en XML se estructuran en *elementos (*como por ejemplo, *body, inertia)* que a su vez pueden contener *atributos (*tales como *name, mass, …)* u otros elementos anidados. Es cada lenguaje basado en XML quien define el nombre y la estructura de los elementos y el tipo de datos de cada atributo.

2.2. Intérpretes o Parsers

Uno de los puntos fuertes de XML es la existencia de *parsers* (intérpretes o compiladores) gratuitos y de gran calidad, implementados principalmente en Java y C++, que simplifican enormemente la lectura y procesado de ficheros de datos. Casi todos los parsers implementan alguna de las dos API's existentes para procesar documentos XML: DOM [9] y SAX [10]. El siguiente fragmento de código Java muestra cómo se procesaría el documento MechML mostrado anteriormente utilizando el parser Xerces-J [11]:

```
import org.apache.xerces.parsers.*;
import org.w3c.dom.*;
...
DOMParser parser = new DOMParser(); // crear objeto Xerces parser
parser.parse(filename);
Element body = parser.getDocument().getDocumentElement();
String name = body.getAttribute("name").getValue();
```

2.3. Validación

Siempre que se lee un fichero de datos es necesario realizar una validación del mismo, verificando que la sintaxis es correcta, que se incluyen todos los elementos necesarios y que los datos son del tipo esperado. XML proporciona mecanismos para realizar esta tarea de modo automático. Una de las formas de hacerlo es describir su estructura empleando el lenguaje

XML Schema [12]. A continuación se muestra cómo se describiría en este lenguaje el fragmento de MechML mostrado anteriormente:

```
<element name="body">
  <sequence>
    <element name="inertia">
      <attribute name="mass" value="double" minInclusive="0"/>
      <attribute name="cogx" value="double"/>
      <attribute name="cogy" value="double"/>
      <attribute name="cogz" value="double"/>
      <attribute name="Ixx" value="double" minInclusive="0"/>
      <attribute name="Iyy" value="double" minInclusive="0"/>
      <attribute name="Izz" value="double" minInclusive="0"/>
    </element>
  </sequence>
  <attribute name="name" value="string"/>
</element>
```

Como puede verse en este ejemplo, XML Schema permite describir la estructura de los elementos y el tipo de dato esperado en cada atributo. MechML ha sido descrito utilizando este lenguaje, y el documento resultante, que constituye la verdadera definición del lenguaje, se ha publicado en Internet (http://lim.ii.udc.es/mechml/MechML.xsd). Puesto que un documento MechML debe hacer referencia a esta URL en su cabecera, el parser que lo procesa puede validar su contenido de forma totalmente automática, y en caso de detectar errores mostraría mensajes descriptivos que permiten detectar fácilmente las causas.

En ciertas ocasiones el lenguaje que se está definiendo tiene reglas que no pueden ser expresadas en XML Schema, pero existen otros sistemas, como The Schematron [13], que permiten completar la definición del lenguaje.

2.4. Herramientas de utilidad

La facilidad para procesar y validar documentos no es la única ventaja de MechML; debido a la gran implantación de XML en Internet existen multitud de utilidades que facilitan el trabajo con información codificada en este formato. A continuación se comentan brevemente aquellas utilidades que resultan más interesantes para trabajar en el campo de la simulación de sistemas multicuerpo:

1) Software de edición diseñado específicamente para trabajar con XML y XML Schema, con capacidad para generar documentación HTML automáticamente [14].
2) XSLT (Extensible Stylesheet Language Tranformation [15]): se trata de un lenguaje interpretado sencillo pero potente que permite transformar la estructura de un documento XML. Esto simplifica la actualización de documentos desde versiones antiguas y la programación de traductores entre lenguajes derivados de XML.
3) Data Binding: existen herramientas que, a partir de un documento XML Schema que contiene la descripción de un lenguaje derivado de XML (tal como MechML), generan

código fuente Java o C++ construyendo una estructura de clases que replica la estructura del lenguaje, simplificando la programación de aplicaciones que trabajen con ese lenguaje.

4) Bases de datos adaptadas para dar soporte a XML [16]: permiten crear bibliotecas de modelos de mecanismos expresados en lenguaje MechML.
5) Resource Description Framework (RDF [17]) es una técnica estándar para describir recursos disponibles en Internet. Al utilizar este sistema para describir documentos MechML (tema, autor, palabras clave, …) se puede facilitar la catalogación y búsqueda automática de modelos de mecanismos.

3. ESTRUCTURA DE DATOS

Una vez vistas las ventajas de emplear la tecnología XML para definir un lenguaje de modelado de mecanismos, de describirá con más detalle el lenguaje MechML utilizando como ejemplo el mecanismo biela-manivela mostrado en la figura 1, que ha sido modelado en coordenadas naturales [19].

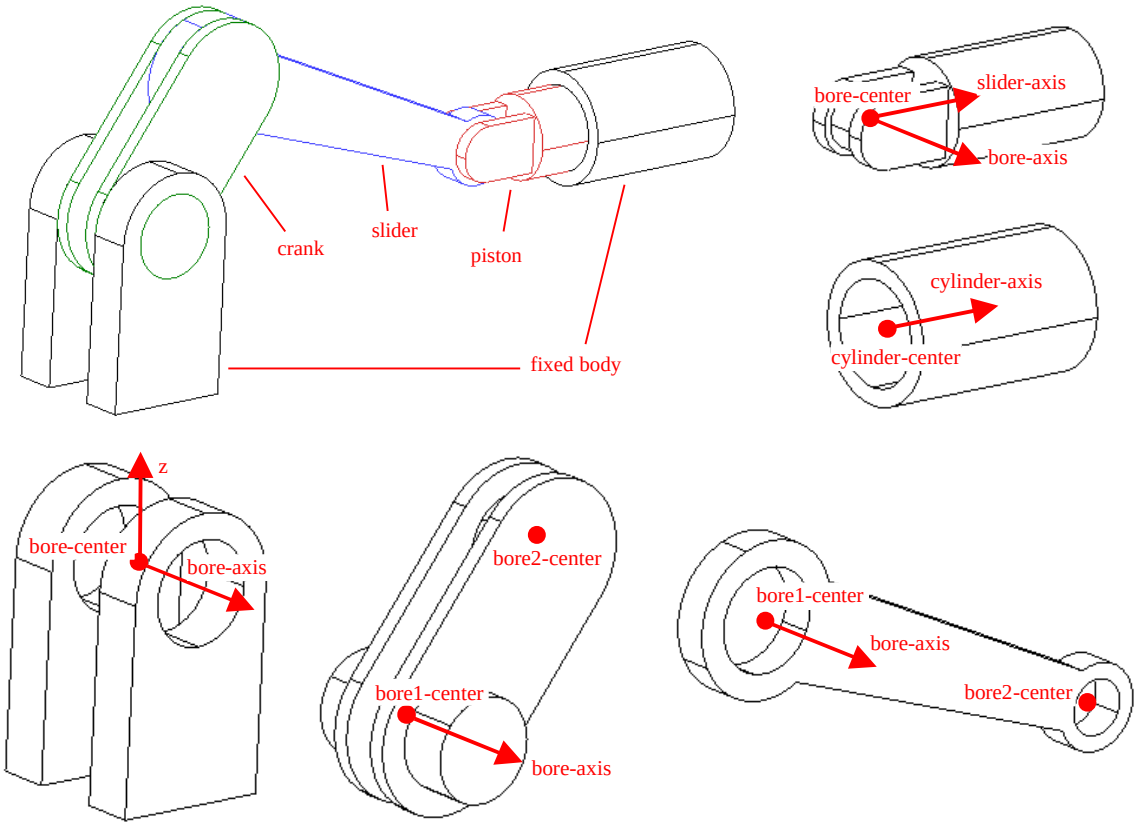

Figura 1. Mecanismo biela-manivela modelado en coordenadas naturales

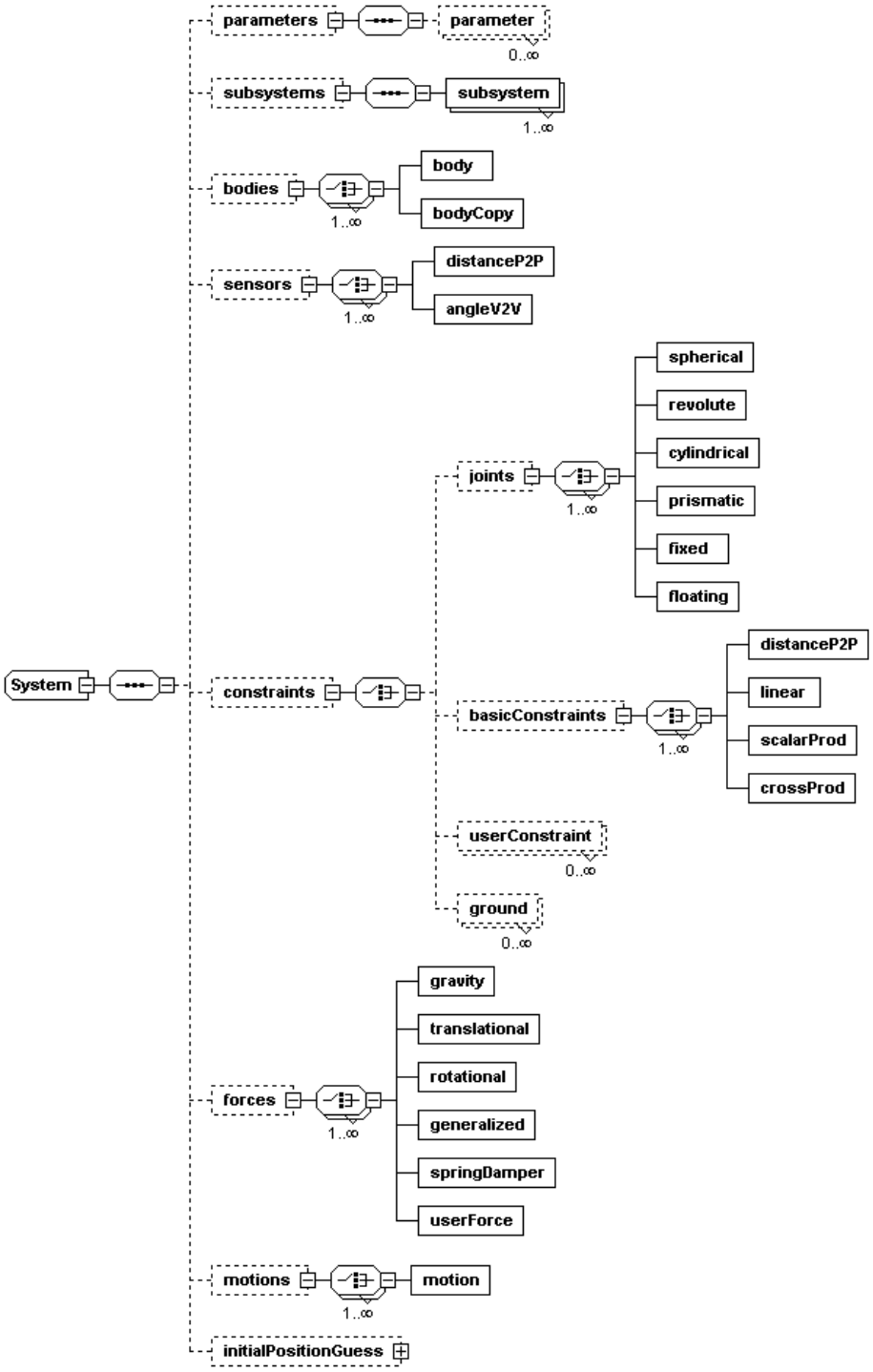

Figura 2. Estructura de datos de MechML: elemento *system*

Un documento MechML se compone de dos elementos: *system*, que describe el sistema multi-cuerpo, y *analysis*, que describe el análisis que se desea realizar sobre el mismo. Las figuras 2 y 3 muestran la estructura de estos elementos. Estas figuras han sido generadas automáticamente desde una herramienta de edición de XML Schemas [18].

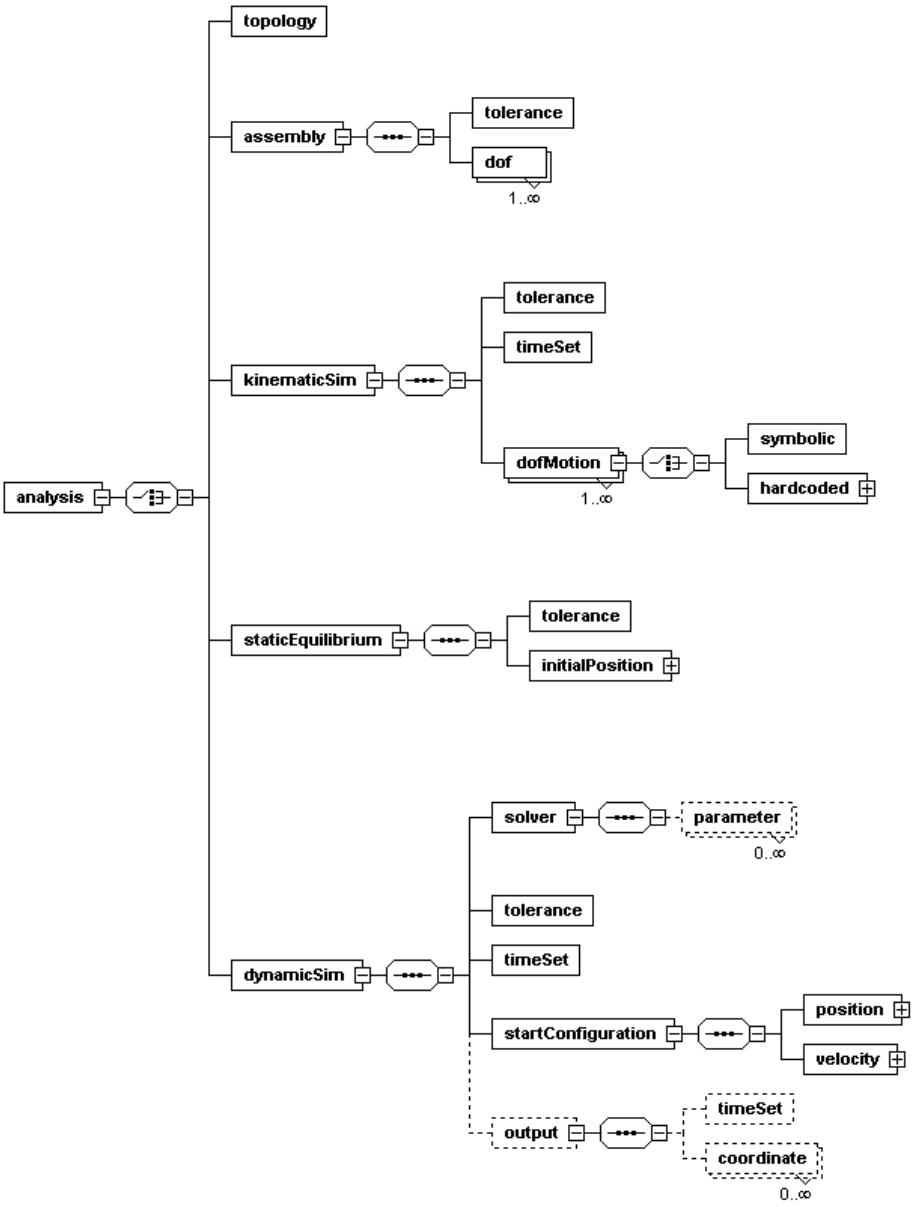

Figura 3. Estructura de datos de MechML: elemento *analysis*

3.1. Descripción del sistema multi-cuerpo

El elemento *system* contiene toda la información necesaria para definir el sistema multi-cuerpo, agrupada en diferentes secciones. Los elementos *body* contienen información sobre cada sólido: nombre, geometría, propiedades inerciales y coordenadas empleadas para definir su posición. Puesto que el mecanismo ha sido modelado en coordenadas naturales, se definen los puntos y vectores utilizados:

```
<body name="slider">
  <graphics format="VRML" source="slider.wrl"/>
  <point  name="center_bore1" x="0.0" y="0" z="0"/>
  <point  name="center_bore2" x="0.045" y="0" z="0"/>
  <vector name="bore_axis" x="0" y="0" z="1.0"/>
  <inertia
    mass="3.2537E-5"
    cogx="0.051858"
    cogy="0.0"
    cogz="0.0"
     Ixx="2.99396E-9"
     Iyy="5.88586E-8"
     Izz="6.13103E-8"/>
</body>
```

Si se necesita medir ángulos o distancias para definir restricciones sobre ellos o simplemente para monitorizarlos, éstos pueden ser definidos en el apartado *sensors*:

```
<angle name=″crank_angle″
  fromVector=″fixedBody:v1″
    toVector=″crank:bore_axis″/>
```

En el apartado *constraints* se modelan las restricciones del mecanismo. Éstas se han agrupado en dos tipos: pares cinemáticos (*joints*) y restricciones básicas (*basicConstraints*). También es posible fijar sólidos (*ground*) o añadir restricciones definidas por el usuario (*userConstraint*).

```
<revolution name="slider-crank">
  <body ref=″slider″>
    <axisPoint ref=″center_bore1″/>
    <axisVector ref=″bore_axis″/>
  </body>
  <body ref=″crank″>
    <axisPoint ref=″center_bore2″/>
    <axisVector ref=″bore_axis″/>
  </body>
</revolution>
```

En el elemento *forces* se describen las fuerzas que actúan sobre los sólidos (gravedad, fuerzas y momentos aplicados, …) y en *motions* los movimientos guiados. Al igual que

ocurría con las restricciones, aquí también se pueden añadir fuerzas o movimientos definidos por el usuario.

MechML también está diseñado para soportar modelos paramétricos. Por ejemplo, si se modela una suspensión de automóvil, se pueden definir las dimensiones y propiedades inerciales de los sólidos utilizando expresiones simbólicas que dependan de ciertos parámetros, que previamente se habrán declarado en la sección *parameters*. Mediante el elemento *subsystem* se puede reutilizar el modelo como parte integrante de otros mecanismos, dando a sus parámetros valores concretos:

```
<subsystem name=”front_suspension” source=”McPherson.xml”>
  <parameter name=”L”         value=”0.450”/>
  <parameter name=”stiffness” value=”0.140”/>
  <parameter name=”damping”   value=”0.0023”/>
</subsystem>
```

3.2. Descripción del análisis

Un buen lenguaje de modelado de mecanismos también debe proporcionar medios para definir el análisis que se desea realizar sobre un determinado sistema. En MechML, el elemento *analysis* contiene esta información: es posible definir análisis topológicos, problemas de posición, simulaciones cinemáticas y dinámicas, y cálculo de posiciones de equilibrio estático.

```
<analysis outputFile="biela-manivela.out.xml">
  <kinematicSim>
    <tolerance rel="1E-3" abs=”1E-4”/>
    <timeSet start="0" end="2.0" step="0.01"/>
    <dofMotion coordinate="crank_angle">
      <symbolic expr="3.14*t"/>
    </dofMotion>
  </kinematicSim>
</analysis>
```

Se está trabajando en una ampliación del lenguaje MechML que permita modelar también los resultados que se desea obtener de los análisis, para facilitar el intercambio y comparación de los mismos.

4. GENERACIÓN DE FICHEROS DE DATOS DESDE ENTORNOS CAD

Escribir a mano el modelo en formato MechML de un mecanismo con un elevado número de elementos y pares puede resultar una tarea bastante tediosa y propensa a errores. Sería deseable poder realizar el modelo en un entorno visual y amigable, como se hace en las herramientas de simulación comerciales. Actualmente todos los paquetes CAD/CAE de gama media-alta pueden comunicarse con otras aplicaciones mediante entornos de computación distribuida (generalmente CORBA o DCOM), lo que permite desarrollar software que

aproveche o extienda sus capacidades a la medida del usuario. Puesto que estos paquetes incorporan módulos de diseño de mecanismos, es posible escribir rutinas que accedan a su estructura de datos y extraigan la información necesaria para generar el archivo MechML con el modelo del mecanismo.

Esta técnica de generación automática de ficheros MechML desde herramientas CAD/CAE ha sido implementada por los autores en el paquete I-DEAS mediante un programa Java llamado Ideas2MechML: el usuario utiliza el módulo de diseño de mecanismos de I-DEAS para modelar la geometría 3D de los sólidos y ensamblarlos añadiendo pares cinemáticos, y a continuación ejecuta el programa Ideas2MechML que, accediendo a la información contenida en I-DEAS, genera el fichero MechML con la definición del mecanismo. Entonces el usuario puede simular el mecanismo en un solver externo programado por él mismo o por terceros.

La versión actual de Ideas2MechML tiene limitaciones: sólo exporta definiciones de sólidos y pares cinemáticos. Las fuerzas, movimientos guiados y la definición del análisis deben añadirse al fichero MechML manualmente. Aún así, el sistema permite un ahorro de tiempo considerable. En el futuro está previsto ampliar MechML para soportar elementos flexibles y utilizar Ideas2MechML para realizar análisis FEM semi-automáticos sobre éstos.

Como parte de este trabajo también se está desarrollando un prototipo de traductor ADAMS-MechML en lenguaje Java.

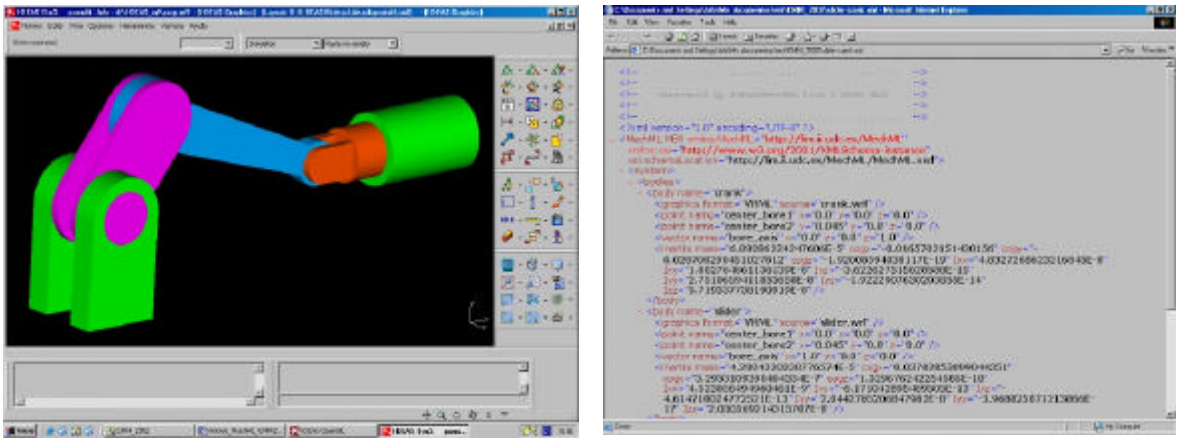

Figura 4. Mecanismo modelado en I-DEAS y archivo MechML generado automáticamente

5. EJEMPLO DE IMPLEMENTACIÓN: SOFTWARE JAVA Y MATLAB

Los lenguajes basados en XML, como MechML, se utilizan como formato estándar para definir, almacenar e intercambiar datos. Si los datos están en un formato estándar siempre se pueden convertir al formato particular que utilice una aplicación concreta.

Aprovechando la facilidad de procesar documentos basados en XML, se pueden desarrollar programas que traduzcan el formato estándar –en este caso MechML– a cualquier otro formato de datos particular.

A modo de ejemplo y con carácter didáctico, se va a describir con un cierto detalle la

implementación de un programa que convierte datos escritos en un dialecto 2-D de MechML al formato particular de una aplicación docente escrita en Matlab, llamada *mech2d*, que realiza diversos tipos de análisis cinemático de mecanismos planos.

El programa *mech2d* es de libre distribución y se puede obtener en la siguiente dirección de Internet: http://mat21.etsii.upm.es/mbs/matlabcode/. En esa misma dirección se puede obtener la aplicación didáctica desarrollada para mostrar la utilidad y la facilidad de implementación de un lenguaje estándar como MechML aunque por motivos históricos o de otra índole disponga de un formato de datos propietario.

El programa *mech2d* consiste simplemente en una función de MATLAB a la que se le pasa como argumento el nombre del archivo de MATLAB en el que definen las estructuras que contienen los datos sobre los elementos y los pares que constituyen el mecanismo a analizar.

A continuación se muestra la definición de un cuadrilátero articulado según el formato particular de *mech2d* basado en MATLAB:

```
ELEMENT(1).nombre = 'ground';
ELEMENT(1).coor = [0,0; 2,0];

ELEMENT(2).nombre = 'barra1';
ELEMENT(2).coor = [0,0; 2,0];

ELEMENT(3).nombre = 'barra2';
ELEMENT(3).coor = [0,0; 2,0];

ELEMENT(4).nombre = 'barra3';
ELEMENT(4).coor = [0,0; 2,0];

JOINT(1).nombre = 'par1';
JOINT(1).tipo = 'revolute';
JOINT(1).elem1 = 1;
JOINT(1).elem2 = 2;
JOINT(1).puntosComp = [1,1];

JOINT(2).nombre = 'par2';
JOINT(2).tipo = 'revolute';
JOINT(2).elem1 = 2;
JOINT(2).elem2 = 3;
JOINT(2).puntosComp = [2,1];

JOINT(3).nombre = 'par3';
JOINT(3).tipo = 'revolute';
JOINT(3).elem1 = 3;
JOINT(3).elem2 = 4;
JOINT(3).puntosComp = [2,1];

JOINT(4).nombre = 'par4';
JOINT(4).tipo = 'revolute';
JOINT(4).elem1 = 4;
```

```
JOINT(4).elem2 = 1;
JOINT(4).puntosComp = [2,2];
```

A continuación se da una breve explicación del fichero anterior. Como se puede observar, los elementos o sólidos rígidos se definen en la estructura ELEMENT por medio de un nombre –*ELEMENT.nombre*– y las coordenadas locales de sus puntos –*ELEMENT.coor*–. Con la estructura *JOINT* se definen tanto los pares cinemáticos como las variables –distancias y/o angulos– que el usuario quiera introducir para la definición del mecanismo.

Todos los pares tiene un nombre –*JOINT.nombre*– y un tipo de par –*JOINT.tipo*–, y dependiendo del tipo se necesitan unos datos u otros.

- Par de revolución (*revolute*). Se necesitan los dos elementos que liga el par –*JOINT.elem1* y *JOINT.elem2*– y los puntos compartidos de cada elemento en la numeración local –*JOINT.puntosComp*–.
- Par prismático (*prismatic*). Se necesitan los dos elementos que liga el par y los tres puntos necesarios para generar las restricciones debidas al par en numeración local –*JOINT.prodVect* y *JOINT.prodEscal*–.
- Par Engranaje (*gear*). Hay que dar los números de los dos ángulos que están ligados, la constante que los relaciona y el ángulo inicial –*JOINT.gAndrap*–.
- Par Piñón-Cremallera (*rackAndPinyon*). Se deben definir los números de la distancia y el ángulo que están relacionados, la constante correspondiente y el valor inicial de la distancia –*JOINT.gAndrap*–.

Otra información necesaria para cualquier análisis cinemático es la posición inicial de los puntos en coordenadas globales, las velocidades y/o aceleraciones según los grados de libertad, y los puntos –con numeración global– que deben dibujarse unidos mediante líneas la representación gráfica. La matriz *POINTS* tiene tres columnas: la primera es el número del punto según la numeración global, y la segunda y la tercera son las coordenadas *x* e *y* de los puntos. En las estructuras *ind* y *indv* se almacena la variable independiente que controla el movimiento del mecanismo, las variables se numeran empezando por los coordenadas de los puntos, después los ángulos variables y por último las distancias variables. En este caso la variable 5 sería la coordenada *x* del punto global 3. En las estructuras *posind* y *velind*, se añade el valor o los valores que van a tomar las variables en las distintas posiciones de la simulación.

```
% Initial position

POINTS=zeros(4,3);
POINTS(:,1)=[1:4]';

POINTS(1,2) = 0;
POINTS(1,3) = 0;
POINTS(2,2) = 2;
POINTS(2,3) = 0;
POINTS(3,2) = 0;
```

```
POINTS(3,3) = 2;
POINTS(4,2) = 2;
POINTS(4,3) = 2;

% Position

ind=[5];
posind=[ind sin(0:pi/24:6*pi) ];

% Velocity

indv=[5];
velind=[indv 1 ];

% Drawing parameters
dists=[ 1 3; 3 4; 4 2];
```

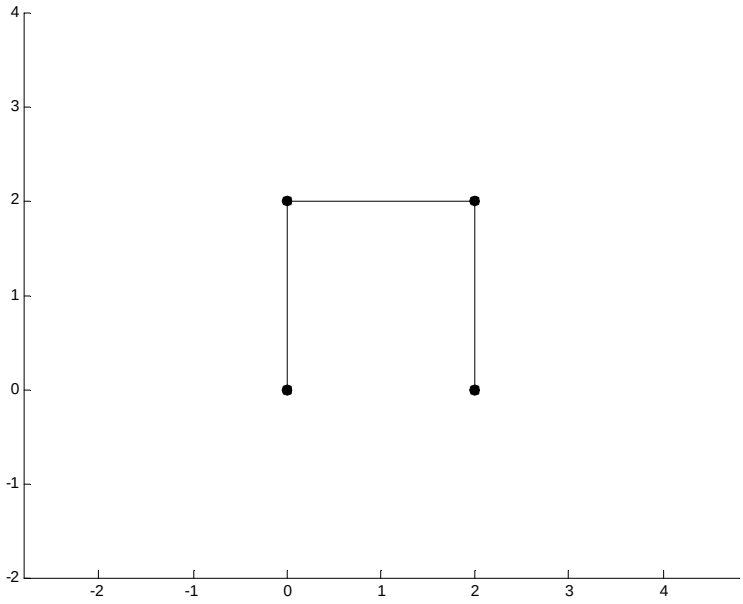

Figura 5. Representación gráfica de Matlab del cuadrilátero articulado definido.

Como se ha podido observar, este formato de datos es propio del analizador *mech2d*, y es muy dificil de exporta y generalizar. MechML hace fácil crear un filtro para pasar datos de un formato estándar al formato específico de *mech2d*.

5.1 Software Java: Mbs2dmlParser

El lenguaje MechML ha sido desarrollado para definir sistemas multi-cuerpo de tres dimensiones. Como en este ejemplo se trabaja con mecanismos planos y un analizador

cinemático, se ha utilizado un dialecto o versión reducida de MechML, ya que hay una serie de datos que no son necesarios, bien para la definición de mecanismos planos (vectores unitarios, pares esféricos, pares cilíndricos...), bien para unos análisis exclusivamente de tipo cinemático (matrices de inercias, masas, fuerzas, momentos, centros de gravedad. ...). Por tanto el programa *Mbs2dmlParser* contrastará los archivos XML con un archivo *Schema* más reducido –*mbs2dML.xsd*–, que se puede obtener en la siguiente dirección de Internet: http://mat21.etsii.upm.es/mbs/schema/mbs2d/mbs2dML.xsd/. El programa *Mbs2dmlParser*. escrito en Java, utiliza la siguiente función:

```
public static void main(String[] args) {
    if (args.length > 0) {
      String filePath = args[0];
      Mbs2dmlParser parser = new Mbs2dmlParser(filePath);
      parser.setContentHandler(new MatlabContentHandler());
      parser.parse();
    }
  }
```

Lo primero que hace la función *main()* es llamar al constructor de la clase pasándole como parámetro el *path* del archivo XML que se quiere traducir a MATLAB. Después se le asigna como *ContentHandler* (interface definida por el programa de traducción con los métodos que recorrerán la estructura en memoria para extraer la información sobre cada tipo de elemento del lenguaje: *body*, *joint*, ...) un objeto de la clase *MatlabContentHandler* y a continuación se llama al método *parse* de la clase *Mbs2dmlParser*.

En el constructor de la clase *Mbs2dmlParser* se crea un objeto de la clase *DOMParser*, utilizando la librería Xerces-J, que lee y crea en memoria la estructura de datos del archivo XML que se quiere analizar. Esta estructura de datos está accesible a través de la variable *docElement*, que es una variable miembro de *Mbs2dmlParser*.

```
public Mbs2dmlParser(String aFilePath) {
    try {
      filePath = aFilePath;
      DOMParser parser = new DOMParser();
      …
      parser.setErrorHandler(this);
      parser.parse(filePath);
      docElement = parser.getDocument().getDocumentElement();
    } catch (Exception e) {
      e.printStackTrace(System.err);
    }
  }
```

Se ha remarcado el método *parse* porque no se debe confundir este método con el método *parse* de *Mbs2dmlParser*. El método *parse* de *DOMParser* analiza el archivo XML y crea la correspodiente estructura en memoria. Posteriormente, el método *parse* de *Mbs2dmlParser*

analiza dicha estructura y genera el archivo de MATLAB con el formato de *mech2d* antes descrito.

Con el método *parse* de *Mbs2dmlParser* se explora qué tipo de estructuras se tienen en memoria y en consecuencia se hacen llamadas a las funciones de *MatlabContentHandler*, que son las que se encargan de crear y generar el archivo de MATLAB correspodiente.

La clase *MatlabContentHandler* deriva de la clase *DefaultContentHandler* que a su vez implementa la interface *ContentHandler*. Dicha interface declara los métodos necesarios para poder traducir las diferentes estructuras que hay en MechML (bodies, joints,...) y la clase *DefaultContentHandler* permite derivar de ella otras clases que pueden redefinir algunos de los métodos de la interface *ContentHandler*, sin necesidad de tener que implementar la totalidad de los métodos. Por tanto si se quisiera traducir a otro formato de datos, lo que habría que hacer sería crear otra clase que derivara de *DefaultContentHandler*.

El archivo de definición del cuadrilátero articulado producido por el programa Java anterior, ya en formato XML, sería tal como se indica a continuación.

```
<bodies>
  <body name="ground">
   <point name="p1" x="0" y="0"/>
   <point name="p2" x="2" y="0"/>
  </body>
  <body name="barra1">
   <point name="p1" x="0" y="0"/>
   <point name="p2" x="2" y="0"/>
  </body>
  <body name="barra2">
   <point name="p1" x="0" y="0"/>
   <point name="p2" x="2" y="0"/>
  </body>
  <body name="barra3">
   <point name="p1" x="0" y="0"/>
   <point name="p2" x="2" y="0"/>
  </body>
 </bodies>
 <joints>
  <joint name="par1" type="revolute">
   <body ref="ground"><point ref="p1"/></body>
   <body ref="barra1"><point ref="p1"/></body>
  </joint>
  <joint name="par2" type="revolute">
   <body ref="barra1"><point ref="p2"/></body>
   <body ref="barra2"><point ref="p1"/></body>
  </joint>
  <joint name="par3" type="revolute">
   <body ref="barra2"><point ref="p2"/></body>
   <body ref="barra3"><point ref="p1"/></body>
  </joint>
  <joint name="par4" type="revolute">
```

```
   <body ref="barra3"><point ref="p2"/></body>
   <body ref="ground"><point ref="p2"/></body>
  </joint>
 </joints>
 <initialPositionGuess bodyRef="ground">
  <guess coordinate="ground:p1:x" value="0"/>
  <guess coordinate="ground:p1:y" value="0"/>
  <guess coordinate="ground:p2:x" value="2"/>
  <guess coordinate="ground:p2:y" value="0"/>
  <guess coordinate="barra1:p2:x" value="0"/>
  <guess coordinate="barra1:p2:y" value="2"/>
  <guess coordinate="barra2:p2:x" value="2"/>
  <guess coordinate="barra2:p2:y" value="2"/>
 </initialPositionGuess>
 <startConfiguration>
  <position>
   <tolerance rel="1.e-05" abs="1.e-05"/>
   <dof coordinate="barra1:p2:x" value="sin(0:pi/24:6*pi)"/>
  </position>
  <velocity>
   <dof coordinate="barra1:p2:x" value="1"/>
  </velocity>
 </startConfiguration>
```

6. LIMITACIONES

MechML no es por el momento un lenguaje completo, sino más bien un prototipo en evolución que necesita ser ampliado y mejorado en gran medida. A continuación se enumeran las limitaciones más importantes de la versión actual:

1) Sólo soporta la modelización de mecanismos mediante coordenadas naturales.
2) No se contemplan elementos flexibles.
3) En el lenguaje actual no se definen unidades, suponiéndose que las unidades de las magnitudes que se especifican son coherentes entre sí. Esto puede originar problemas por ejemplo al utilizar modelos definidos como subsistemas en sistemas de mayor complejidad.
4) No se pueden introducir cambios de configuración en el transcurso del análisis, es decir, no se pueden modelar restricciones o pares cinemáticos que aparezcan o desaparezcan en el tiempo.
5) No se ha previsto por el momento la forma de definir análisis más específicos, como los problemas de contacto, los impactos, el juego y rozamiento de Coulomb en los pares, etc.

7. CONCLUSIONES

En esta comunicación se ha presentado MechML, un prototipo de lenguaje de modelado de sistemas multi-cuerpo cuya característica fundamental es ser un lenguaje basado en XML. La tecnología XML aporta a MechML ventajas significativas frente a

otros lenguajes de modelado de mecanismos: sencillez y claridad, facilidad para procesar y validar documentos, utilización de estándares, integración con las tecnologías de Internet, etc.

Todas estas características hacen de MechML un buen candidato para convertirse en el lenguaje estándar de modelado de mecanismos. Sin embargo MechML todavía no es un lenguaje maduro, necesita ser ampliado y mejorado.

REFERENCIAS

[1] ISO, ISO 10303: 1994, Industrial Automation Systems and Integration – Product Data Representation and Exchange – STEP Part 105: Kinematics, URL: http://www.iso.ch/cate/cat.html.

[2] M. Otter, H. Elmqvist and F. Cellier, "Modeling of Multibody Systems with the Object-Oriented language Dymola", *Nonlinear Dynamics*, **9**, pp. 91-112 (1996)

[3] H. Elmqvist, S. E. Mattsson and M. Otter, "Modelica: The New Object-Oriented Modeling Language", The 12th European Simulation Multi-conference, Manchester, U.K.

[4] World Wide Web Consortium, *eXtensible Markup Language (XML)*, 2000, URL: http://www.w3.org/XML/.

[5] World Wide Web Consortium, *Mathematical Markup Language (MathML)*, 2001, URL: http://www.w3.org/Math.

[6] Chemical Markup Language (CML), URL: http://www.xml-cml.org.

[7] Geographical Markup Language (CML),

[8] NASA, *Instrumental Markup Language (IML)*, 2001, URL: http://pioner.gsfc.nava.gov/public/iml/.

[9] World Wide Web Consortium, *Document Object Model Specification (DOM)*, 2000, URL: http://www.w3.org/TR/REC/REC-DOM-Level1/.

[10] David Megginson, *Simple API for XML (SAX)*, 2000, URL: http://www.megginson.com/SAX/.

[11] Apache Software Foundation, Xerces-J, 2000, URL: http://xml.apache.org/xerces-j/.

[12] World Wide Web Consortium, *XML Schema Specification*, 2001, URL: http://www.w3.org/TR/schema/.

[13] R. Jelliffe, *Schematron*, 2001, URL: http://www.ascc.net/xml/resouce/schematron/.

[14] XML Software, URL: http://www.xmlsoftware.com.

[15] World Wide Web Consortium, *Extensible Stylesheet Language Tranformation*, 2001, URL: http://www.w3.org/TR/xslt/.

[16] XML:DB Initiative: Enterprise Technologies for XML Databases, URL: http://www.xmldb.org.

[17] World Wide Web Consortium, *Resource Description Framework (RDF) Model and Sysntax Specification*, 1999, URL: http://www.w3.org/TR/REC-rdf-syntax/.

[18] XML Spy Integrated Development Environment, URL: http://www.xmlspy.com.

[19] J. García de Jalón and E. Bayo, *Kinematic and Dynamic Simulation of Multibody*

Systems, Springer-Verlang, 1994.