Polyplan

Upgraded implementation of the kinematic modeling tool Polyplan, using Java and XML

Yoash Levron & Albert Schoute
University of Twente
Department of Computer Science

Abstract

Polyplan is a configuration tool for visualization of moving objects in 3D-space. It uses a modeling language (formatted in XML), by which the geometric configuration and the kinematic properties of general world models can be specified. Objects are represented by convex polyhedrons of arbitrary complexity. Objects move according to configuration parameters that define the kinematic transformations of their coordinate frames.

Introduction to kinematic modeling

In a world model that contains moving objects like vehicles, manipulator links, obstacles, etc. many parameters are needed in order to describe the configuration completely. However, whereas most parts are considered as rigid bodies, it is common practice to separate out the set of (dynamic) parameters that actually vary the configuration and to leave aside all (static) parameters that remain fixed. These dynamic parameters are termed ‘configuration parameters’.

Assume a world configuration depends upon m configuration parameters that may vary, each within some appropriate range. Then, all configurations can be represented as points in an m-dimensional space, name ‘configuration space’ [Lozano-Perez 83].

The placement of any rigid body depends upon some subset of the configuration parameters. Fixed objects such as immobile obstacles do not depend on any parameter. Moveable bodies will depend upon one or more parameters. A manipulator link connected by a joint to some other object depends on the joint parameter in addition to the parameters of the object to which it is attached. In figure 1 an example object configuration model is shown with some typical cases. We consider single objects being parts that behave as rigid bodies. In the table below the dependency of objects on the configuration parameters is shown.

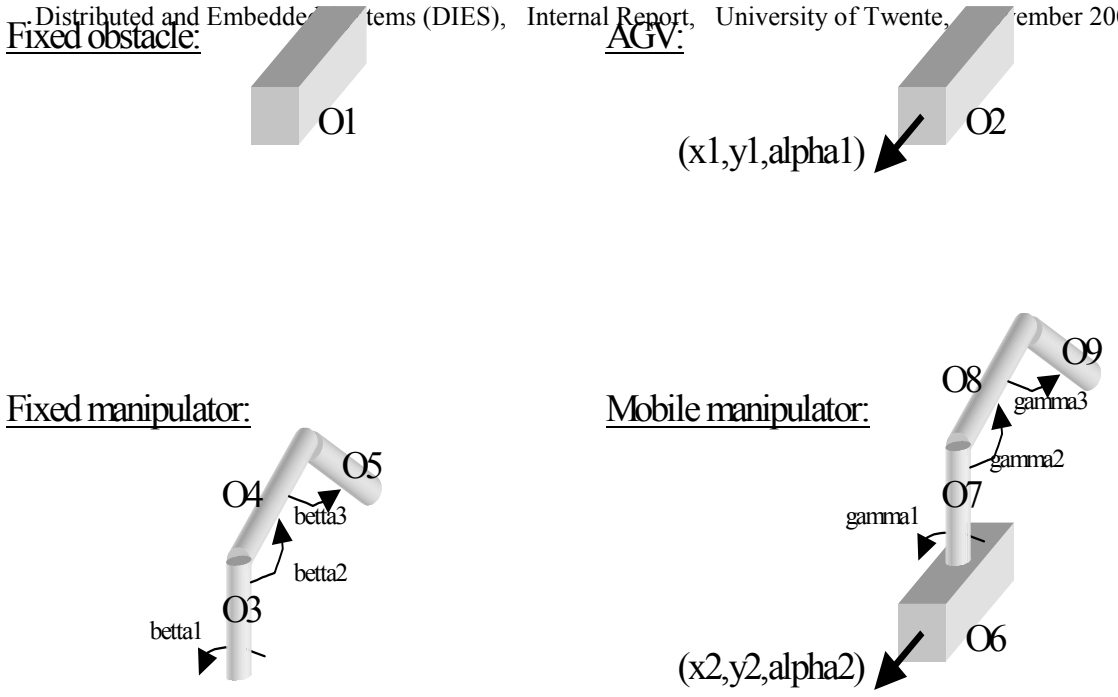

Figure 1.
Typical object configuration model dependent upon configuration parameters x1,y1,alpha1,x2,y2,alpha2,beta1,beta2,beta3,gamma1,gamma2,gamma3.

	X1	Y1	Alpha1	X2	Y2	Alpha2	Beta1	Beta2	Beta3	Gamma1	Gamma2	Gamma3
O1	-	-	-	-	-	-	-	-	-	-	-	-
O2	+	+	+	-	-	-	-	-	-	-	-	-
O3	-	-	-	-	-	-	+	-	-	-	-	-
O4	-	-	-	-	-	-	+	+	-	-	-	-
O5	-	-	-	-	-	-	+	+	+	-	-	-
O6	-	-	-	+	+	+	-	-	-	-	-	-
O7	-	-	-	+	+	+	-	-	-	+	-	-
O8	-	-	-	+	+	+	-	-	-	+	+	-
O9	-	-	-	+	+	+	-	-	-	+	+	+

Dependency of objects on the configuration parameters.

To each object a coordinate frame can be attached. The shape of the object can be expressed relative to that frame. By means of affine transformations, i.e. translations and rotations, coordinate frames can be expressed relative to each other. In order to fix frames in the configuration space we use a common reference frame attached to the outside world (named the base frame).

We will assume that objects have shapes modeled by convex polyhedrons. Of course, in practice this shape corresponds to some sufficiently accurate enclosure of the real object. A polyhedron will be specified by its set of boundary vertices. Each vertex will be described as a position vector in the object frame. By using frame transformations these vertex positions are (given a certain configuration) easily mapped to the base frame (or any other frame).

Configuration modeling

The world model and some representation parameters are read from a specification file. An example of such a file is given in the appendix. It reflects a configuration model as shown in figure 1 at the beginning of this report. In fact a kind of modeling language has been defined, that uses an XML syntax.

The specification file defines:

- Constants (using the 'const' elements)
- Configuration parameters (using the 'param' element)
- Coordinate frames, which are possibly relative to each other ('frame' elements)
- Polyhedral objects ('object' element). Objects can be attached to frames. The objects shape is determined by a list of vertexes ('polyform' elements).
- The eye parameters and the starting location of the world origin, which define the way that the world will be viewed on screen.

Frames are defined by supplying arguments (previously defined parameters and constants) to transformation functions (such as the Denavit-Hartenberg transformation).

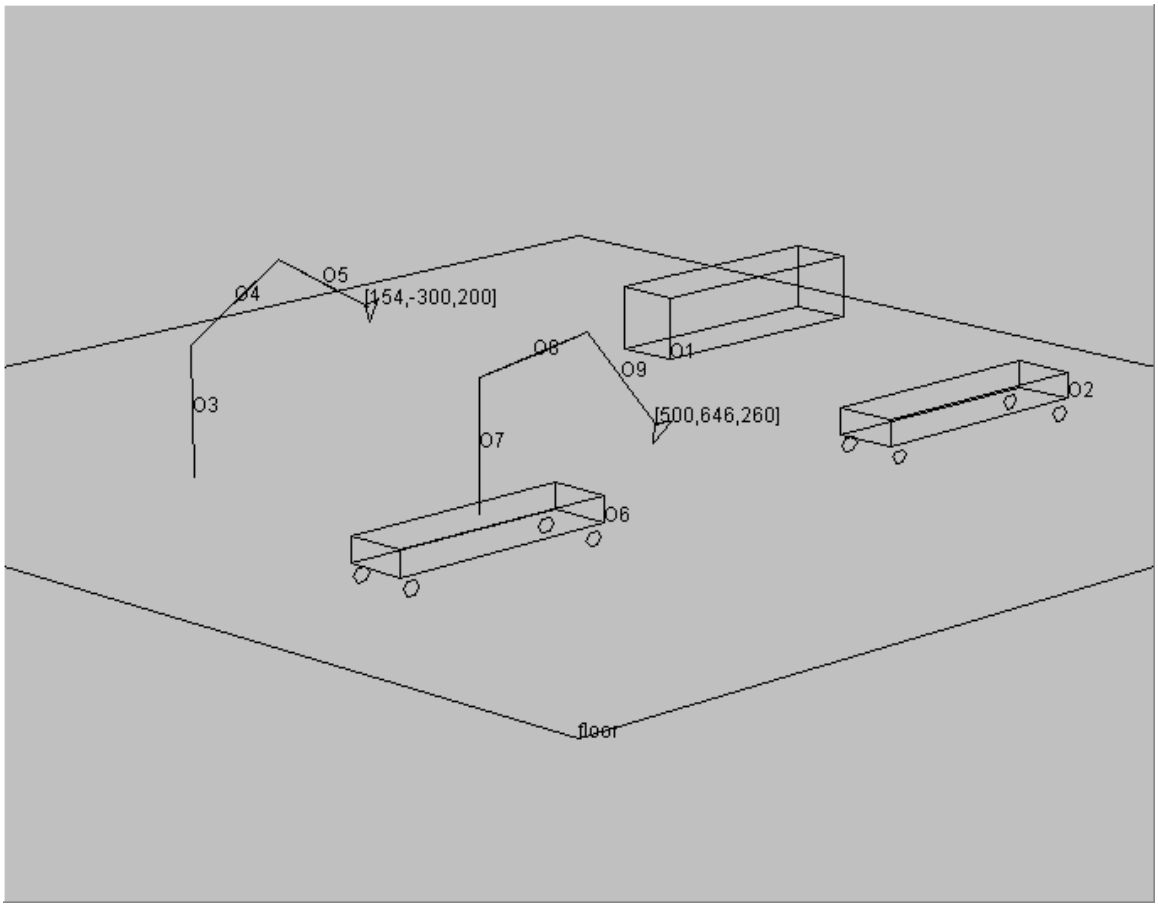

Figure 2. Scene as generated from 'example.xml'

A short explanation about XML

XML is a very general syntax, which is used to define other languages.
The specification syntax is defined in a DTD file (Data Type Definition). The Polyplan program uses the 'polyplan.dtd' file to define its syntax. This file must be in the same directory as the source XML file, otherwise the program will not work.

The XML syntax is built out of elements, which are ordered in a hierarchical structure.
Each XML element can contain other elements, and can contain attributes.
The general syntax of an XML element definition is:

```
<elementName  attributeName=”attributeValue”  attributeName=”attributeValue” … >
                Child Element 1
                Child Element 2
                    …
</elementName>
```

General properties of XML files:

- White spaces, tabs, or new-line characters do not affect the syntax.
- Attributes inside the elements can appear in any order.
- Remarks can appear anywhere in the XML file. Remarks are added by: <!-- This is a remark -->
- Every XML file contains one root Element. All the other elements are contained in that root element. In the Polyplan syntax, the root element will always be called 'polyworld'.

The polyplan specification file syntax

Like we said, the syntax is defined using an XML syntax. For that reason every specification file will begin with the following code:

```
<?xml version='1.0' encoding='US-ASCII'?>
<!DOCTYPE polyworld  SYSTEM "polyplan.dtd">
<polyworld
```

and will end with the code word : </polyworld>

this code defines the file as an XML file , includes the DTD file (which must be in the same directory as the XML file) , and defines the root element – 'polyworld'.

All other element must appear inside the root element, and must be one of the followings: <const> , <param> , <frame> , <polyform> , or <object>.
The elements must appear in that exact order, that means that all the <const> elements will come first, then the <param> elements, and so forth. The <object> elements will appear last. This restriction was added mainly for implementation reasons.

General properties of the Polyplan syntax:

- Each element has a name attribute which identifies it. It is illegal for two elements of the same type to have the same name.
- All identifier names must begin with a letter.
- All degrees are given in radians.
- Numerical values have to be given according to standard real denotation.
- If a Numerical value is followed immediately by an 'R' or 'r' it is interpreted as angle given in degrees and converted into radians. For example: "180R" will be interpreted as the value of pi.

Eye and center definitions

The eye and center are defined as attributes of the <polyworld> element.
And so, their definition must come right after the code word: <polyworld,
at the beginning of the specification file, like so:

```
<polyworld
        centerratiox="0.5"
        centerratioy="0.5"
        eyephi="45R"
        eyetheta="75R"
        eyedistance="10000"
        eyezoom="0.4"
```

(The numerical values here are given as an example).
Those attributes will define the way that the world will be viewed on screen.

centerratiox – this attribute defines the location of the world origin on the screen horizontal axis. A "0" value will define the most left column as the origin location, and "1" will define the most right column as the origin location. This attribute can be omitted, in which case a default value of "0.5" will be taken.
centerratioy – the same as centerratiox on the vertical axis.
eyephi , eyetheta – this values defines the viewing angle.
eyedistance - defines how far is the "eye" from the origin. In general, this should be a large number.
eyezoom – defines the "eye" zoom factor.

Constants

Constants are defined by:

<const name="constName" value="constValue" > </const>

Constants can replace any other value (numerical or not) in the specification file, (except from the eye and center attributes). It is not possible to define a constant by using another constant.

Parameters
Configuration parameters are defined by:

<param name="paramName" min="minValue" max="maxValue" value="initialValue"> </param>

The 'value' attribute can be omitted. In which case the initial value will be taken as the average of the minimum and maximum values.
The configuration parameters can be used only as arguments to frame functions.

Frames
Coordinate frames (to which objects can be attached later) are defined by:

<frame name="frameName" inframe="relativeFrameName">
<frameFunctionName … frame function parameters …> </frameFunctionName>
</frame>

This phrase defines a frame 'frameName' relative to another frame. The latter frame must have been defined before. In case the inframe attribute is omitted, the frame is taken relative to the base frame. Frames are specified be means of a transformation function supplied with the proper arguments. Transformation functions must be known to the system. A few standard transformation functions are built in the system. The arguments of the frame functions can be numerical values, constants, or configuration parameters. Presently the following frame functions are recognized:

<pose2d x="xValue" y="yValue" alpha="alphaValue"> </pose2d>

'pose2d' defines a translation of the frame in the x-y plane, and rotation 'alpha' around the z-axis. (this function is a special case of 'pose3d' with z="0" beta="0" gamma="0").

<pose3d x="xValue" y="yValue" z="zValue"
alpha="alphaValue" beta="betaValue" gamma="gammaValue"> </pose3d>

'pose3d' defines a general 3D specification with Carthesian translation vector (x,y,z) and Euler ZYZ rotation angles (alpha, beta, gamma).

<dhtrans alpha="alphaValue" a="aValue" d="dValue" theta="thetaValue">
</dhtrans>

'dhtrans' defines a frame transformation according to the Denavit-Hartenberg convention. In principle, any of the four argument may be filled in as configuration parameters. However, generally only one argument is taken as a variable. Namely, 'theta' in case of rotational link, and 'd' in case of a prismatic link. Robots consisting of pairwise connected links can be modeled easily by using the 'dhtrans' transformation function.

Polyforms

Before introducing an object, first its polyhedral shape must be specified. Multiple objects can be modeled according to the same specified form. A polyhedral shape is defined by:

```
<polyform  name="formName">
        <vertex  x="xValue"  y="yValue"  z="zValue"  mark="mark"> </vertex>
        <vertex  x="xValue"  y="yValue"  z="zValue"  mark="mark"> </vertex>
        <vertex  x="xValue"  y="yValue"  z="zValue"  mark="mark"> </vertex>
                        ….
</polyform>
```

each 'vertex' phrase defines a vertex of the form. All vertices together determine the polyhedral shape of the form. The 'mark' attribute is an optional single character.
Presently there are two possible values to 'mark':

- "c" or "C" – the coordinates of the vertex will be printed in the graphical output.
- "n" or "N" – the object's name will be printed in the graphical output.

Objects

Objects are defined by:

```
<object   name="objectName"  form="formName"  frame="frameName"
          scale="value"  xscale="value"  yscale="value"  zscale="value"
          x="value"  y="value"  z="value"
          nearto="nearObjectName"  virtual="yesOrno">
</object>
```

Apart from the 'name' and 'form' attributes all the other attributes are optional.

'form' –
Defines the polyhedral form which makes this object.

'frame' –
If defined, the object is attached to this frame (which must have been defined before). If not defined, the object will be attached to the base frame (world coordinates).

'scale', 'x/y/zscale' -
The object size may be scaled by a uniform scaling factor in all dimensions, and by separate scaling factors in the x,y,z dimensions. If both are defined, the 'scale' value will multiply the 'xscale', 'yscale' and 'zscale' values. If omitted, scaling values are taken as "1". A negative scale will make the 'mirror' form of the form that makes this object.

'x/y/z' –
Defines the objects location in the coordinate frame. If omitted, the objects location is (0,0,0).

Note that those attributes enable objects with the same shape to be put in different places in different sizes, using only a single polyhedral form.

The 'nearto' and 'virtual' attributes affect only the distance analyzer, and so, have any meaning only when the 'distance analysis' option is selected. In the current version of the program, the distance analysis is not working so well, so we recommend not to use it.

Some documentation about the Java/C implementation

This software was implemented using both the C and Java languages. The Java code is responsible for the user interface and the parsing of XML files. On the other hand, the C code is responsible for the computational part of the software.

The link between the Java and C languages is implemented using the JNI system (Java Native Interface). Explanations about JNI can be found at:
http://java.sun.com/docs/books/tutorial/native1.1/concepts/index.html.
The JNI provides a way to integrate an existing source code into Java applications. This integration is achieved by using a few simple steps:
1) The Java class, which will use the native functions, is written and made (compiled). In the Polyplan software this class is called cInterface.
2) An .h file is automatically generated from the Java class, using the javah.exe program. This .h file contains the signatures of the C functions (which are used by the Java code). In the Polyplan software this file is called cInterface.h.
3) The implementation of the 'h' file is written in the proper C file (cInterface.c in Polyplan). The functions in cInterface.c use the functions that are implemented by the C files of the computational part.
4) All the C files are compiled and linked as a Dynamic Link Library. In Polyplan this file is called cInterface.dll.

The global variables in the C-code are referred as 'the C variables'. These variables reside in the memory, which is reserved to the .dll file.
The old C code used functions that draw to screen, using graphichs.h. The Polyplan implementation overrides these functions. Instead of drawing shapes on screen, these functions will create a data-base, (part of the C variables), which will save all the shapes to-be-drawn on screen. Some of the Java functions in the cInterface class are able to access this data-base in order to draw those shapes (in the proper Java window).
To understand this drawing process better, lets look at an example. The old Polyplan software used the 'line' function, declared in graphics.h, to draw lines on screen. The new Polyplan software overrides this function. A call to the new 'line' function will add a component to new lines-data-base (the array 'lineArr', declared in graphics.c). Notice that the original code, which calls the 'line' function, was not changed. In order to actually draw the line on screen, the Java cInterface class contains functions that are able to access this line-data-base. An example of such a function is 'getLineX1', which retrieves the horizontal value of the first coordinate of a specific line.

The Java code is divided into classes. Since polyplan was made using the Jbuilder software, some of these classes are machine generated. Polyplan contains three important classes:

cInterface

This class contains all the Java methods, which are implemented using the C language. The class contains methods, which are divided by their functionality:

- Methods for manipulating the C variables. Such a method for example is 'setConfigurationParameter', which causes a manual change of a configuration parameter value.
- Methods for accessing the to-be-drawn shapes data base. Such a method is the one we mentioned before – 'getLineX1'.
- Methods which are used for updating the C-variables from an XML file. These methods replace the old 'ConfigScene' C-function (in config.c). All the names of these functions starts with the prefix 'init'. An example of such a method is 'initAddParam', which adds a new configuration parameter to the system.

Parser

This class implements the parsing of XML files. The important method is 'run', which will parse the XML file. If the file contains errors, they will be printed in the proper window (which is given in the parser's class constructor). If the file is valid, the C variables will be updated to match the XML file. The parsing is implemented using the common and simple SAX parser. The implementation of this parser is given in the SAXParserFactory class. A very good explanation about the SAX parser can be found at: http://java.sun.com/xml/jaxp/dist/1.1/docs/tutorial/overview/index.html.

Frame1

This class implements the main frame, and therefore most of the user interface. This class contains many event handlers for the various actions that the user can make. Big parts of this class are machine generated by Jbuilder. There are two visual components in this class, which were manually generated (not created by Jbuilder). One is the window which contains the configuration parameters. This window could not be automatically generated because the number of configuration parameters it contains is dynamically varying. The second manually made visual component is the panel that shows the picture of the scene. This panel was made manually because there is no standard Java-Swing component, which can handle drawings. A good explanation of how to manage non-standard drawings in Java can be found at the following URL:
http://java.sun.com/docs/books/tutorial/uiswing/painting/index.html

The other classes are Frame1_aboutbox and Polyplan. These classes are completely machine generated and contain no important details.

The software components

Here is a list of the source files needed to make the Polyplan software:

Java classes:

- polyplan.java – contains a machine generated Java class which uses an instance of the Frame1 class. This class resembles to the 'main' function.
- Frame1.java – contains the Frame1 class.
- Frame1_Aboutbox.java – contains the machine generated Frame1_AboutBox class, which is used by Frame1.
- Parser.java – contains the Parser class.
- cInterface.java – contains the cInterface class.

C-files:

- cInterface.c – contains the implementation of all the C-Java interface functions. The file cInterface.h, which is included here, is machine generated using javah.exe. The file jni.h, which is included, is part of the JNI system.
- Config.c – contains functions which were used by the old Polyplan program to config the scene using a .geo file. Most of the functions here are not used in the current version of Polyplan.
- Frame.c – contains functions to manipulate frames.
- Goalwalk.c – was used by the old version of Polyplan. Most of the functions are not being used in the current version.
- Graphics.c – contains the implementation of the graphics data-base and drawing functions, as explained before. Typical functions of this file are line, circle, and outtext.
- Kinema.c - contains most of the computational implementation of Polyplan.
- Linalg.c – contains typical linear algebra functions.
- Maximize.c – also implements some of the computational part, but most of the functions are not used in the current version.
- Polydist.c – responsible for the distance computation part, which is not working so well on the correct version of Polyplan.
- Polyplan.c – this file contains the old C 'main' function, which is not needed at all on the current version of Polyplan. However, it is very comfortable to use it for debugging.
- Scenery.c – contains functions that draws the 3D world on the screen. The most important function is 'DrawScene'.
- Visual.c – also contains drawing functions.

Other-files:

- C header (.h) files.
- polyplan.dtd – the XML data type definition file, which defines the Polyplan XML syntax.
- Example XML files.
- PolyplanGuide.doc – this document, in Word format.

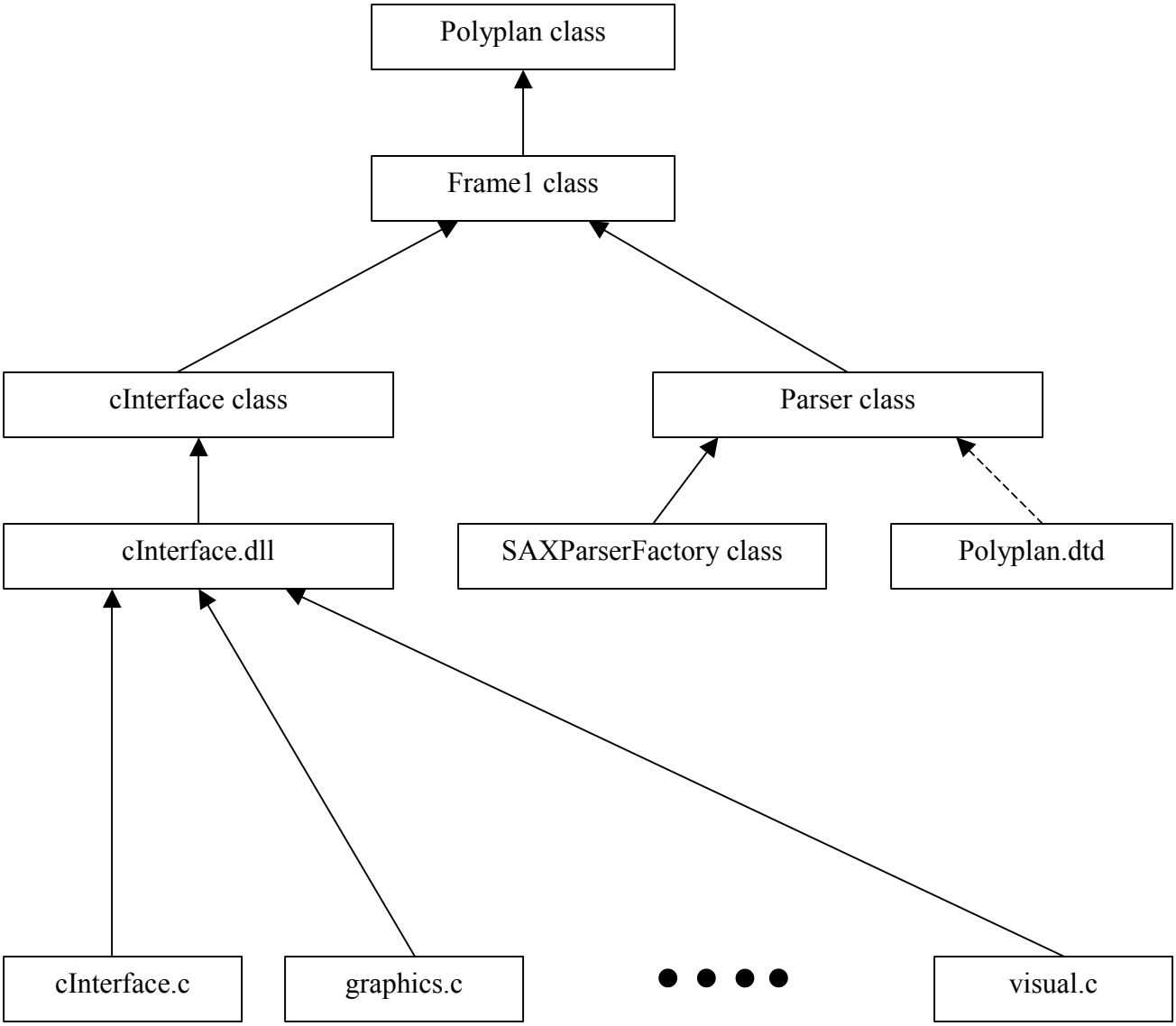

Figure 3. This figure shows the relations between the various software components. The various components are ordered hierarchically, every level of the hierarchy is using the lower levels.

References

[Craig86]
Craig J.J., Introduction to robotics, Mechanics and Control, Addison-Wesley 1986.

[Lozano-Perez83]
Spatial planning: a configuration space approach, IEEE Transactions on Computers, volume C32, no. 2 pp. 108-120.

The URL http://java.sun.com contains good documentations, examples, and tutorials about Java and XML.

Appendix 1 – specification file "example.xml"

```
<?xml version='1.0' encoding='US-ASCII'?>
<!DOCTYPE polyworld SYSTEM "polyplan.dtd">

<polyworld
        centerratiox="0.5"
        centerratioy="0.5"
        eyephi="45R"
        eyetheta="75R"
        eyedistance="10000"
        eyezoom="0.4"
        >

        <!--    notice the order of keywords:
                consts,parameters,frames,polyforms,objects:   -->

        <const name="xmani" value="500"> </const>
        <const name="ymani" value="-300"> </const>


        <param name="x1"            min="-1000"     max="1000"      value="-500">   </param>
        <param name="y1"            min="-1000"     max="1000"      value="300">    </param>
        <param name="alpha1"        min="-180R"     max="180R"      value="90R">    </param>
        <param name="beta1"         min="-180R"     max="180R"      value="-180R">  </param>
        <param name="beta2"         min="-90R"      max="0R"        value="-30R">   </param>
        <param name="beta3"         min="0R"        max="90R"       value="60R">    </param>
        <param name="x2"            min="-1000"     max="1000"      value="500">    </param>
        <param name="y2"            min="-1000"     max="1000"      value="300">    </param>
        <param name="alpha2"        min="-180R"     max="180R"      value="90R">    </param>
        <param name="gamma1"        min="-180R"     max="180R">                     </param>
        <param name="gamma2"        min="-180R"     max="0R"        value="-30R">   </param>
        <param name="gamma3"        min="-90R"      max="90R"       value="60R">    </param>


        <frame name="agv">
                <pose2d x="x1"  y="y1"  alpha="alpha1"> </pose2d>
        </frame>
        <frame name="fixman">
                <pose2d x="xmani"  y="ymani"  alpha="0"> </pose2d>
        </frame>
        <frame name="link03" inframe="fixman">
                <dhtrans alpha="0"  a="0"  d="200"  theta="beta1"> </dhtrans>
        </frame>
        <frame name="link04" inframe="link03">
                <dhtrans alpha="-90R"  a="0"  d="0"  theta="beta2"> </dhtrans>
        </frame>
        <frame name="link05" inframe="link04">
                <dhtrans alpha="0"  a="200"  d="0"  theta="beta3"> </dhtrans>
        </frame>
        <frame name="grip1" inframe="link05">
                <pose3d x="200"  y="0"  z="0"  alpha="0"  beta="0"  gamma="0">  </pose3d>
        </frame>
        <frame name="carrier">
                <pose2d x="x2"  y="y2"  alpha="alpha2"> </pose2d>
        </frame>
        <frame name="link07" inframe="carrier">
                <dhtrans alpha="0"  a="0"  d="260"  theta="gamma1">  </dhtrans>
        </frame>
        <frame name="link08" inframe="link07">
                <dhtrans alpha="-90"  a="0"  d="0"  theta="gamma2">  </dhtrans>
        </frame>
        <frame name="link09" inframe="link08">
                <dhtrans alpha="0"  a="200"  d="0"  theta="gamma3">  </dhtrans>
        </frame>  <frame name="grip2" inframe="link09">
                <pose3d  x="200"  y="0"  z="0"  alpha="0"  beta="0"  gamma="0"> </pose3d>
        </frame>
```

```
<polyform name="square" >
        <vertex  x="-0.5"  y="-0.5"  z="0">                    </vertex>
        <vertex  x="-0.5"  y="0.5"   z="0">                    </vertex>
        <vertex  x="0.5"   y="-0.5"  z="0">                    </vertex>
        <vertex  x="0.5"   y="0.5"   z="0"      mark="N">      </vertex>
</polyform>
<polyform  name="box" >
        <vertex  x="-0.5"  y="-0.5"  z="-0.5">                 </vertex>
        <vertex  x="-0.5"  y="-0.5"  z="0.5">                  </vertex>
        <vertex  x="-0.5"  y="0.5"   z="-0.5">                 </vertex>
        <vertex  x="-0.5"  y="0.5"   z="0.5">                  </vertex>
        <vertex  x="0.5"   y="-0.5"  z="-0.5">                 </vertex>
        <vertex  x="0.5"   y="-0.5"  z="0.5">                  </vertex>
        <vertex  x="0.5"   y="0.5"   z="-0.5"   mark="N">      </vertex>
        <vertex  x="0.5"   y="0.5"   z="0.5">                  </vertex>
</polyform>
<polyform name="wheel" >
        <vertex  x="0"     y="0.25"  z="0.35">                 </vertex>
        <vertex  x="0"     y="0.25"  z="-0.35">                </vertex>
        <vertex  x="0"     y="0.5"   z="0">                    </vertex>
        <vertex  x="0"     y="-0.25" z="0.35">                 </vertex>
        <vertex  x="0"     y="-0.25" z="-0.35">                </vertex>
        <vertex  x="0"     y="-0.5"  z="0">                    </vertex>
</polyform>
<polyform name="Xlink" >
        <vertex  x="0"     y="0"     z="0">                    </vertex>
        <vertex  x="0.5"   y="0"     z="0"      mark="N">      </vertex>
        <vertex  x="1"     y="0"     z="0">                    </vertex>
</polyform>
<polyform name="Zlink" >
        <vertex  x="0"     y="0"     z="0">                    </vertex>
        <vertex  x="0"     y="0"     z="0.5"    mark="N">      </vertex>
        <vertex  x="0"     y="0"     z="1">                    </vertex>
</polyform>
<polyform name="gripper" >
        <vertex  x="0"     y="0"     z="0"      mark="C">      </vertex>
        <vertex  x="0.2"   y="0.2"   z="0">                    </vertex>
        <vertex  x="0.2"   y="-0.2"  z="0">                    </vertex>
</polyform>


<object name="floor"   form="square"  scale="2000"> </object>
<object name="O1"      form="box"
        xscale="400"          yscale="100"          zscale="100"
        x="-700"  y="-350"    z="50"> </object>
<object name="O2"      form="box"    frame="agv"
        xscale="100"          yscale="400"          zscale="40"
        x="0"     y="0"       z="40"  nearto="floor"> </object>
<object name="wheel11"  form="wheel"  frame="agv"
        scale="30"  x="50"  y="-180"  z="0"  virtual="yes"> </object>
<object name="wheel12"  form="wheel"  frame="agv"
        scale="30"  x="50"  y="180"   z="0"  virtual="yes"> </object>
<object name="wheel13"  form="wheel"  frame="agv"
        scale="30"  x="-50"  y="-180"  z="0"  virtual="yes"> </object>
<object name="wheel14"  form="wheel"  frame="agv"
        scale="30"  x="-50"  y="180"   z="0"  virtual="yes"> </object>
<object name="O3"  form="Zlink"  frame="link03"  scale="-200"
        nearto="floor"> </object>
<object name="O4"  form="Xlink"  frame="link04"  scale="200"> </object>
<object name="O5"  form="Xlink"  frame="link05"  scale="200"> </object>
<object name="gripper1"  form="gripper"  frame="grip1"  scale="100"> </object>
<object name="O6"  form="box"    frame="carrier"
        xscale="100"  yscale="400"  zscale="40"
        x="0"  y="0"  z="40"  nearto="floor"> </object>
<object name="wheel21"  form="wheel"  frame="carrier" scale="30"
        x="50"  y="-180"  z="0"  virtual="YES"> </object>
<object name="wheel22"  form="wheel"  frame="carrier" scale="30"
        x="50"  y="180"  z="0"  virtual="YES">  </object>
```

```
        <object name="wheel23"  form="wheel"  frame="carrier" scale="30"
                x="-50"  y="-180"  z="0"  virtual="YES"> </object>
        <object name="wheel24"  form="wheel"  frame="carrier" scale="30"
                x="-50"  y="180"  z="0"  virtual="YES"> </object>
        <object name="O7"  form="Zlink"  frame="link07" scale="-200" > </object>
        <object name="O8"  form="Xlink"  frame="link08"  scale="200"> </object>
        <object name="O9"  form="Xlink"  frame="link09"  scale="200"> </object>
        <object name="gripper2"  form="gripper"  frame="grip2"  scale="100"> </object>
</polyworld>
```