A Language for Constructive
Parametric Solid Modelling

Lluis Solano
Pere Brunet

Report LSI-94-43-R
A Language for Constructive Parametric Solid Modelling

Lluis Solano, Pere Brunet

September 29, 1994

Universitat Politècnica de Catalunya
Departament de Llenguatges i Sistemes Informàtics
Secció d'Informàtica Gràfica
Av. Diagonal 647 planta 8 (ETSEIB)
08029 Barcelona. Spain

E-mail: solano@lsi.upc.es
E-mail: brunet@lsi.upc.es

Abstract

The report discusses proposed solutions for constraint-based modelling, with special emphasis on constructive approaches. A new constructive scheme that overcomes a number of the present limitations is proposed. It is based on a non-evaluated, constructive solid model. The proposed approach supports instantiation of pre-defined models, parametric geometric operations in 1D, 2D and 3D, variable topologies, and operations with structural constraints. The EBNF specification of the model definition language is presented and discussed through several examples.

Keywords: Geometric modelling, solid modelling, CAD, constraint-based modelling, parametric design.

1 Introduction

Design process plays an important role in industrial production of new products. Recently, many tools including Computer-Aided design have been developed for improving the productivity of design process.

The term CAD is related to the use of computers as an aid to the entire design process, including creation, modification, and visualization of designed parts. Traditional CAD-
systems focus at the explicit object being designed. The construction process and the
relations between the involved elements are not reflected in the final design.

Present CAD systems can be very useful for the design and representation of specific, final
products. However, they still have a number of drawbacks that must be solved in order
to use them in practical design applications. There is a need for facilities for conceptual
design, and better tools for the generation of the design are required. On the other hand,
the use of previous designs in a new one is not always supported. Finally, in most CAD
systems it is necessary to define the exact size and location of every geometric element
and/or part. This is of course too rigid in many applications, where the interest of the
user is to generate a prototype model from a number of shape specifications, and then
adjust it through shape modification tools in order to optimize the performances of the
product.

It is necessary to use efficient tools that increase the flexibility in the design process, by
defining the geometry of the elements and the relations among them without specifying
the set of concrete dimensions or locations of the elements that are involved in a design.

Parametric CAD systems can design general objects that represent a family of different
objects, sharing the same topological constraints but having different geometry [CFV88].
Given a set of specific parameters, particular components of the family are obtained.
Parametric models store the geometry of the object together with variable dimension
parameters [RSV89], [Rol91a]. Parametric design increases the flexibility in the design
process, by defining the geometry and geometric constraints without specifying the set of
concrete dimensions of the object.

With the use of geometric constraints in parametric design it is possible to specify the organ-
ization of a design. In this sense it is easy to generate design variations using constraints
[Rol91a]. Constraint-based design is aimed at representing and capturing designer's in-
tend. It allows to design generic objects more than explicit ones and ever a family of
designs instead a single one.

Parametric design technique [Rol91a] is becoming a useful methodology for conceptual
design, tolerance analysis, efficient design of families of parts, representation of standard
parts and features in libraries, kinematics simulations, and assemblies design. Several
parametric design approaches have been proposed that will be reviewed in section 2.

The present paper proposes a constructive definition of the object model that overcomes a
number of the present limitations of parametric systems. It supports instantiating of pre-
defined models, parametric geometric operations in 1D, 2D and 3D, variable topologies
and operations with structural constraints. Next section discusses some of the well-known
approaches for parametric design, focusing on constructive schemes and comparing their
performances and limitations. The proposed constructive scheme is then presented in
section 3. Section 4 deals with the proposed definition language. Section 5 presents and discusses user interaction and model generation. Section 6 deals with the presented Internal Model Representation. Finally section 7 discusses and present several connected examples.

2 Proposed schemes and Constructive Approaches in Parametric Design

Roller [Rol91a] proposes a classification of parametric design approaches into variants programming, numerical constraint solvers, expert systems and constructive schemes. In variants programming, the user must write a procedure in a certain programming language whenever he wants to define a parametric object. Variants programming is widely used in CAD systems through macro definition languages, mainly for the definition of parts libraries. The main drawback of this approach however is that it is not practical for non-expert users.

The numeric constraint solver approach or algebraic method, translates all the geometric constraints into a set of equations. The shape of a part can be defined and modified based on a set of characteristic points of the geometric model. An algebraic system of equations relates the characteristic points to the constraints. The geometry of a specific part is computed by solving the system of equations with an iterative numerical method.

Several versions of this approach have been proposed [HiB78] [LiG81] [LiG82] [LeA85] [Nel85] [Owe91] [Ser91]. In this method the number of equations and variables grows fast with the number of geometric elements and constraints involved. The numerical resolution needs good starting points in order to converge and the computational cost is expensive.

From the user's point of view, it is difficult to express a design in terms of a system of equations that reflects the relationship between the geometric entities of the design. Furthermore, the user doesn't have any feedback about inconsistent shapes or unexpected solutions.

Another approach in parametric design, is the use of an expert system in order to create a geometric model based on a set of constraints. The constraints are thus expressed as rules or predicates [Ald88] [Sun88] [SAK90] [Yak90]. Given the constraints and the starting points, an inference engine is used to determine sequentially the unknown positions of the geometric elements.

This method allows to use more complex constraints. However it needs a large number of predicates for simple shapes [Yak90]. The expert system approach doesn't seem to support
incremental design. Moreover, it doesn’t support cyclic constraints and it is expensive in memory and computations.

In the constructive approach the sequence of interaction performed by the user in order to define an object is recorded by the system. The design sequence is described interactively and graphically by the user. The previous works using this method can be classified into two main groups according to the way the user’s actions are recorded. In the first group the interactive design generates a procedural description of the modelling operations of an object [RBN89] [Emm89] [Emm90] [Kon90] [Rol90] [Rol91b]. In the second group the design sequence is used for the management of a data structure that reflects the relationships between the geometric elements [Ros86] [CFV88] [GZH88].

There are many aspects that characterize a constructive method, however the most significant ones are:

- Whether the method generates a procedure or manages directly a data structure.
- The dimensionality of the method (2D/3D).
- The existence of mechanisms to detect inconsistencies.
- The need to specify all constraints.
- The possibility of instantiating other objects during the building of a new object (instantiating previous models).
- The existence of a mechanism to validate the parameters of an object or to fix their range of validity.
- The ability to detect and support topological changes.
- Whether it is possible to parametrize modelling operations.

Table 1 discusses these aspects in relation to the most representative methods in the constructive approach.

<table>
<thead>
<tr>
<th>Aspect</th>
<th>CPV88</th>
<th>EMM90</th>
<th>GZH88</th>
<th>Kon90</th>
<th>RBN15</th>
<th>Rol91</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of method</td>
<td>Structural</td>
<td>Procedural</td>
<td>Structural</td>
<td>Procedural</td>
<td>Structural</td>
<td>Structural</td>
</tr>
<tr>
<td>Dimensionality</td>
<td>2D</td>
<td>3D</td>
<td>3D</td>
<td>2D/3D</td>
<td>2D</td>
<td>2D</td>
</tr>
<tr>
<td>Inconsistency detection</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>All constraints</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Instantiates previous model</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Validates parameters</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Parameterizes modelling operations</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Detects topological changes</td>
<td>?</td>
<td>?</td>
<td>Yes</td>
<td>?</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

The result of the interaction with the user, together with an accurate set of parameters can be used in order to generate any object of a specific family. In general, this approach
is based on the idea of programming with example [GPG90]. The user defines an example which is recorded by the system. It is thus necessary to keep the history of the design [YKH87] and this can be performed by recording the modelling operations and geometric constraints.

The design of an object is incremental. The design is a sequence of states that converges towards the final object. Every state is characterized by a set of geometric elements and geometric constraints, it is relations that must be satisfied. The evolution between states is done through modelling operations and by adding new geometric constraints.

The designer uses modelling operations and constraints while he is building the object in a natural way. The constructive approach preserves the user's traditional working environment and encourages his ability to decompose a problem into subproblems. In this sense it can be said that the system assists the user in the design of a solution. The geometric elements on which the system is based are simple user sketches whose exact sizes and positions are not needed. It is only necessary to define constraints. Under this perspective, the system doesn't need expert users and the user interface plays an important role. When the interactive design is finished, it can be immediately evaluated in order to modify the design sequence or to perform a new design of a part. Unfortunately, the classical constructive approach doesn't support circular constraints, that must be solved simultaneously.

3 The Constructive Parametric Solid Model

A constructive parametric solid model (CPSM) can be defined as the procedural description of the sequence of modelling operations and constraints performed by the user during the interactive design of a parametric object. It must be observed that the constructive parametric solid model is transparent to the user; the user simply interacts with the system through the graphical user interface in order to generate a particular object that will become the representative of the parametric family. The CPSM can be considered as a generic model of the whole family of objects. The CPSM is a procedural description with a set of formal parameters. Instances - specific object models - of the CPSM are obtained by fixing the values of its formal parameters. The design process when using a constructive parametric solid model involves the following steps:

- A particular object of the parametric family of objects is designed through a graphical user interface and using available modelling operations and constraints. Dimensions and operation parameters can be either constant, related to other dimensions through constraints, or defined as a function of the formal parameters of the model. Constraints can be introduced at any moment during the design process. The corresponding constructive parametric solid model which tracks the design process is
automatically generated. The constructive parametric solid model is represented by a sequence of statements from a definition language (see section 4). Modelling operations and constraints are expressed through procedure calls in the language. This kind of ERep (Editable Representation) has the advantage of being editable, suitable for storing and transmission, it supports both generic and specific designs, and records the conceptual construction steps [RBN89] [HoJ92]. The formal description of the constructive solid model language in a simple case with a limited set of basic geometric elements is presented in section 4.

- The constructive parametric solid model can be evaluated in order to generate specific objects. Different sets of parameter values generate different specific objects, all of them from the same parametric family.

Therefore, the proposed parametric system consists of two modules:

- The module that generates the constructive parametric solid model depicted by a sequence of available modelling operations and constraints. Figure 1 shows the generic structure of this module. The user chooses the modelling operations and constraints through a graphic user interface. A model generator translates it and generates the CPSM that represent the generic object.

![Diagram](image)

**Figure 1: Generation of the Constructive Parametric Solid Model.**

- The evaluation module which generates specific object models with a particular geometry and topology given a set of parameter values. This module (figure 2) includes a translator that generates an internal model representation. The validation and evaluation of the CPSM is done on this internal model representation model. With specific parameter values, the evaluation module produces a particular object model.
Figure 2: Evaluation of Constructive Parametric Solid Model.

It is possible to define assemblies using instances of previously defined parametric models, which are available through a models library. In addition to the representation of a generic object through modelling operations and constraints, the constructive parametric solid model includes information on the explicit geometry of the element which was generated during its design phase. As a consequence, the system can automatically deal with underconstraint situations.

The domain of the proposed constructive parametric solid model is obviously limited by the power of the underlying language. On the other hand, it presents the following advantages:

- It represents the conceptual construction steps, in an incremental way.
- Intermediate models can be stored in generic object libraries, and they can be instantiated in later designs.
- It uses a scheme based on the design with example. Having always the default geometry of the specific object designed by the user, it automatically supports underconstraint cases.
- The system is structured in three independent modules: the interaction module, the internal model representation, and the validation and evaluation module.
- The generic model CPSM is an editable representation, able for archive and transmission.
- The CPSM has a uniform representation for geometries of different dimensions. It is therefore a global approach involving 2D and 3D parametric design.
- It supports structural parameters, that is, parameters in geometric operations (2D to 2D, 2D to 3D or 3D to 3D operations).
The present approach is in some sense parallel to that of [HoJ92], [RBN89], [Emm90]. Hoffman-Juan presents a general framework for ERp languages in solid modelling, Rossignac et al. and Van Emmerick propose an specific language for the representation of the model. Rossignac et al. proposes to record the parameterized sequence of design operations. However, his approach is oriented to intensional model and feature-based design. Van Emmerick’s approach works directly in 3D, but the user can only interact with a set of characteristic points of the model (geometric tree). Our approach is addressed to the design of generic objects in terms not only of modelling operations but also constraints. The user can interact with the whole geometry of the specific object being designed. The presented model covers both 2D and 3D parametric design.

4 The Constructive Parametric Solid Modelling Language

In this section we present the formal specification of the constructive solid modelling language. A model is defined as a sequence of statements. The model has a name given by an identifier and a list of formal parameters that are used in order to evaluate the model. Every statement can be a modelling operation or a geometric constraint.

As we have said before, we propose the use of a language as a model in the parametric object representation. The language definition is done through the specification of a set of functions and operations that the user can perform. The language definition presented is a subset of a full language that reflects all possible modelling operations and constraints performed by the user.

In the present language definition we have into account the following geometric elements: points, vectors, polygons, objects, models and references. Points and vectors are represented by three real numbers corresponding to the 3D coordinates. A polygon is represented using one point of the polygon and a reference in order to define the spatial situation. A reference is a coordinate reference represented by one point and two vectors, the first one fixes the start point for the two vectors that define the normal direction and the x direction. The term object is related to 3D objects and is referenced by one point of the 3D object.

\[
\text{point: } <x,y,z> \quad x,y,z : \text{float}
\]
\[
\text{polygon: } <r,p> \quad r : \text{reference} \\
\quad p : \text{point}
\]
\[
\text{vectors: } <x,y,z> \quad x,y,z : \text{float}
\]
reference: \langle p, n, v_x \rangle

\begin{align*}
p & : \text{ start point} \\
n & : \text{ vector} \\
v_x & : \text{ X vector direction}
\end{align*}

When appear someone of the previous elements in a design process, each one has associated an identifier that allow his reference in the model definition.

In order to specify functions and operations each design stage is characterized by the geometry and constraints defined. The state specification of a model consist on the existing geometry and defined constraints:

\[
M = \langle G, C \rangle \\
G: \text{ geometry} \\
C: \text{ constraints}
\]

The syntax of a model definition is as follows:

\[
\text{model_definition} = \text{MODEL} \text{ identifier} \left( \text{ formal\_parameter\_list } \right) \text{ return} \text{ type\_id} \\
\text{ "{" statement\_list "}".}
\]

\[
\text{ formal\_parameter\_list } = \text{ parameter} \{ , \text{ parameter} \}.
\]

\[
\text{ statement\_list } = \text{ statement; } \{ \text{ statement; } \}.
\]

\[
\text{ parameter } = \text{ identifier } : \text{ type\_id}.
\]

In a model the geometric elements that are allowed are

\[
\text{ type\_id } = \text{ integer} \\
| \text{ real} \\
| \text{ geometric\_element}.
\]

\[
\text{ geometric\_element } = \text{ point} \\
| \text{ polygon} \\
| \text{ object} \\
| \text{ model} \\
| \text{ reference} \\
| \text{ vector}.
\]
identifier = letter { letter | digit } |
| geometric_element_reference.

geometric_element_reference = point_id |
polygon_id |
object_id |
model_id |
reference_id |
vector_id.

point_id = identifier "."PT point_model.

polygon_id = identifier "."PL polygon_model.

object_id = identifier "."OB object_model.

model_id = identifier.

reference_id = identifier "."RF reference_model.

vector_id = identifier "."VC vector_model.

An statement of the model is a geometric function that performs a operation or a constraint between the existing geometric elements. In model definition operations creating a geometric element as well as instanciating existing previous models with the associated parameters are possible.

statement = assignment | constraint | return.

assignment = identifier ".":=" expression.

expression = (expression) |
| identifier |
| constant |
| unary_op expression |
| expression binary_op expression |
| creation_function.

return = RETURN identifier
The geometric element creation is performed calling an available defined functions that are explained in the following section.

creation_function = identifier "(" parameter_list ")".

collection = identifier ", identifier".

constraint = identifier "(" parameter_list ")".

4.1 Defined functions

The present language definition include following list of functions in order to create the geometric elements involved in a model.

4.1.1 Creation of point 3D

Point3D( x:real, y:real, z:real ) return point
{
 Pre: M=<G,C>, (x,y,z) are the 3D coordinate of a point
 Post: M=<G.(x,y,z),C>

 Creates a point 3D
}

2P_Point_3D( P1:point, P2:point, perc:real) return point
{
 Pre: M=<G, C>, P1 and P2 are the initial reference points in 3D, perc
 is the percentage of the distance P1P2 where the new point will
 be created

 Post: M=<G.(x,y,z), C.(x=x1+perc*(x2-x1),y=y1+perc*(y2-y1),
 z=z1+perc*(z2-z1))>

 Create a 3D point to certain percentage of the distance between two
existing points
}

4.1.2 Creation of polygons

Present definition takes into account two cases for the polygon creation: regular polygons and polygons created by a sequence of 3D points.

The creation of a regular polygon is performed by the following function:

```c
Reg_pol( c:integer, long:real, P:point, n:reference ) return polygon
{
Pre: M = <G, C>,
   c = number of sides,
   long = side length,
   P = center of the polygon (it is a 3D point that belongs to the reference plane),
   n = reference plane

Post: M = <G.(c points belonging to the reference plane, .
   C.(polygon constraint in each one of the points).
   (constraint between every point of the polygon and P)>

between the center point P and each one of the polygon vertex Pi
the constraint defined is:
   vector(P.Pi) perp n
   |P.Pi| = radi
   |P.Pi+1| = long

A regular polygon with c points and long side is created with center in P
in the plane defined by n.
}
```

In order to define a polygon from a sorted point list the following function exist:

```c
Closed_pol( point, point, point,..., n:reference ) return polygon
{Pre: M=<G,C>,
```
point, point, point... = polygon point list that are on
the reference plane
n= reference plane

Post: M=<(G,C). (polygon constraint between the vertexs of the polygon)>

Creates a polygon in the plane referenced by n, from a point list
sequence

The creation of 3D objects is performed by the following function call:

Paral_sweep ( P: polygon, l:real ) return object
{Pre: M=<(G,C>
P= polygon to sweep
l= sweep length, (l <> 0.0)

Post: M=<(G.(vertexs of the new polygon),C.(constraints
between polygon).
constraints between the points of the new polygon)>}

Creates an object by parallel sweep in the normal direction of
the P polygon. The initial polygon is duplicated at the
distance given by l. Constraints between polygons and points
belonging to the same object are created.

As an operation it is also possible to modify an existing object through a sweep operation
on one of the object faces, that is performed as follows:

Mult_sweep( O:object, P: polygon, P’: polygon, l:real) return object
{Pre: M=<(G,C>
P = polygon of the face of G that will support the sweep (P belongs to G)
P’= polygon that will be swept,
for each vertex vertex belongs to P’, vertex belongs to plane(P)
l= sweep length

Post: M=<(G.(geometry once P’is swept), C.(constraints of the duplicated
polygon). (sweep constraints). (object constraints)>

Polygon P’ is swept from the polygon P. The object that belongs
the polygon P is modified according to the sweep of the polygon P'
}

Given two 3D objects, it is possible to perform their logical union with the function call:

Union(01:object, 02:object) return object
{Pre: M=<G,C>
  01= first object
  02= second object

Post: M=<G,C>
  Creates a new object form by 01 and 02
}

4.2 Defined function constraints

In this section we will present the defined function constraints that are used in the CPSM. Function constraints represents the geometric constraints that it is possible to fix between geometric elements involved in a design.

Any explicit constraint defined by the user is translated to a one of the following function calls:

Dist_2P( P1:point, P2:point, d:real)
{
  Pre: M= <G,C>
  P1= first point
  P2= second point
  d = distance between the two points P1 and P2

Post: M= <G,C.(constraint distance between P1P2)>
  The geometry of P1 and P2 has to satisfy the constraint

  \[ d^2 = (x1-x2)^2 + (y1-y2)^2 + (z1-z2)^2 \]

  Creates a constraint between the points P1 and P2
}

The distance constraint allows to perform the coincidences between two points. Coincidences are a particular case of distance. With distance 0 the constraint will be:
Coincident_2P( P1: point, P2: point )
{
  Pre: M=<G,C>
  P1, P2= constrained points

  Post: M=<G,C.(coincidence constraint between P1 and P2)>
  P1.x = P2.x
  P2.y = P2.y
  P3.z = P3.z

  Creates a constraint between the two points that appear as parameters
}

As particular cases it is possible to have one coordinate coincidence between two points:

Coincident_X( P1: point, P2: point, n:reference)
Coincident_Y( P1: point, P2: point, n:reference)
Coincident_Z( P1: point, P2: point, n:reference)

We also have the particular cases of distance in the direction of the coordinate axis. These constraints are:

X_dist( P1:point, P2:point, n:reference, d:real )
{
  Pre: M= <G,C>
  P1= first point
  P2= second point
  d = distance along X axis between P1 and P2
  n = reference plane

  Post: M= <G,C.(point constraint between P1P2)>

  The geometry of P1 and P2 verifies
  d = P1.x-P2.x

  Creates a X distance constraint between points P1 and P2
}

Y_dist( P1:point, P2:point, n:reference, d:real )
{
  Pre: M= <G,C>
P1= first point
P2= second point
d = distance along Y axis between P1 and P2
n = reference plane

Post: M= <G,C.(point constraint between P1P2)>

The geometry of P1 and P2 verifies
d = P1.y-P2.y

Creates an Y distance constraint between points P1 and P2

Z_dist( P1:point, P2:point, n:reference, d:real )
{
Pre: M= <G,C>
P1= first point
P2= second point
d = distance along Z axis between P1 and P2
n = reference plane

Post: M= <G,C.(point constraint between P1P2)>

The geometry of P1 and P2 verifies
d = P1.z-P2.z

Creates a Z distance constraint between the points P1 and P2
}

It is also possible to define a constraint between points involving an angle.

Angle_3P( P1:point, P2:point, P3:point, a:real )
{
Pre: M= <G,C>
P1= center point
P2= first end point
P3= second end point
a= angle value between P1P2 and P1P3

Post: M= <G,C.(point constraint between P1P2P3)>
a= angle(P1P2,P1P3) (anticlockwise)
The geometry of P1, P2, P3 verifies the constraint

We also have the possibility to define a constraint that fixes the angle between two vectors:

\[
\text{Angle\_2dir}(\ v1:\text{vector}, \ v2:\text{vector}, \ a:\text{real})
\]

\{
\begin{align*}
\text{Pre: } & M= <G,C> \\
& v1= \text{first vector} \\
& v2= \text{second vector} \\
& a= \text{angle value between vectors } v1 \text{ and } v2
\end{align*}
\]

\[
\text{Post: } M= <G,C. (\text{constraint between the point that form } v1 \text{ and } v2)> \\
& a= \text{angle}(v1,v2) \text{ (anticlockwise)}
\]

Another useful constraint is the constraint that fixes the coincidence of the normal vector of two polygons:

\[
\text{Equal\_normal}(\ F1:\text{polygon}, \ F2:\text{polygon})
\]

\{
\begin{align*}
\text{Pre: } & M= <G,C> \\
& F1, \ F2= \text{constrained polygons}
\end{align*}
\]

\[
\text{Post: } M= <G,C. (\text{coincidence constraint between the normal vector of } \\
& F1 \text{ and the normal vector of } F2)> \\
& \text{Creates a constraint between the two polygons}
\]

4.3 Auxiliary functions

In this section we explain a set of auxiliary functions that appear in the model in order to inquire element values. Following there is a list of some functions to query geometric values as distance, angle, centroid, plane equation. These values are calculated between geometric elements, mainly points.
Dist( P1:point, P2:point ) return real
{
    Returns the distance between P1 and P2
}

Dist_X( P1:point, P2:point, R:reference ) return real
{
    Returns the X distance between P1 and P2
}

Dist_Y( P1:point, P2:point, R:reference ) return real
{
    Returns the Y distance between P1 and P2
}

Dist_Z( P1:point, P2:point, R:reference ) return real
{
    Returns the Z distance between P1 and P2
}

Angle_3P( P1:point, P2:point, P3:point ) return real
{
    Returns the angle value between P1P2 and P1P3 (anticlockwise)
}

Angle_vect( v1:vector, v2:vector ) return real
{
    Returns the angle value between v1 and v2 (anticlockwise)
}

Centroid( P:polYGON) return point
{
    Returns the centroid of the indicated polygon
}

Plane_3P( P1:point, P2:point, P3:point) return reference
{
    Defines a reference plane and the coordinate system in it
    P1P2= X axis with P1 as point (0,0,0),
    P3= third point in order to define the XY plane
    reference vector = P1P2 x P1P3
}
Ref_plane( P: polygon ) return reference
{
    Creates a reference from a polygon

    reference : n=vector normal plane, P=centroid, vector_x=first edge
}

Ref_vector( P1: point, P2: point ) return vector
{
    Defines a vector between the point P1 (start) and P2 (end)
}

Normal_pcl( P: pol_id ) return vector_id
{
    Returns the normal vector of the given polygon
}

DEF_REF() return reference
{
    Returns the default reference coordinate system
}

XY_REF() return reference
{
    Returns a reference such that its XY plane of the default reference coordinate system
}

XZ_REF() return reference
{
    Returns a reference such that its XZ plane
}

YZ_REF() return reference
{
    Returns a reference such that its YZ plane
}
5 User Interaction and Model Generation

The user interacts with the system through a graphical user interface. The user interface is an independent process that manages and parses the operations provided by the user. Furthermore, the user interface manages the visualization of the model in progress. From the user operations the system generates the statements of the representation model through the following steps:

- The user interacts with the already designed geometry, and defines new operations and constraints.
- The system automatically generates the language description of the new designed features, and assigns symbolic names to any new geometric element.
- In parallel, an specific object model is stored with the explicit geometry given by the user interaction, in order to visualize it during the next interactive design steps.

The set of basic geometric elements that can be used in modelling operations and constraints include 0D elements (points), 1D elements (lines and edges), 2D elements (planes, polygons) and 3D elements (polyhedra and objects 3D). All of them must be instantiated and are parametrically defined. They act as primitives within the final constructive parametric solid model. Modelling operations can either keep the dimension of the operands (for instance, in the case of boolean set operations between 3D elements) or increase it (like in sweep operations that transform a 2D element onto a solid). Modelling operations can be parametric operations, that is, the result depends not only on the operands but also on the value of a number of formal parameters.

Both the interaction process and the structure of the constructive parametric solid model can be clarified through the design of a simple object. A family of 'L-shaped' solids is to be generated. Being $L$ the edge size of the square faces, figure 3, the heigh of the part must be $2 \times L$ and its length must be $2 \times L \times F$, $F > 0.5$ (figure 3-a). Different specific objects from the same parametric family can be obtained by giving particular values to the parameters $L$ and $F$. The design process starts by instantiating a square (one of the supported 2D parametric objects) and performing a sweep operation in order to generate a parametric prism with a general heigh $H$. The system automatically generates the corresponding constructive parametric solid model of the prism according to the following sequence of user's actions:

- select 'define new model' and enter its name.
- select 'define a regular polygon'.

20
- enter the parameters involved.
- select 'generate solid by parallel sweep' and define the parameters as:
  select the polygon just generated
  define the value of the sweep as a parameter of the model.

The result generated by the system is:

```plaintext
MODEL square_prism (L,H) return object
{
  square_prism.P01 := Reg_pol (4, L, point(0., 0., 0.))
  square_prism.OB1 := Paral_sweep (square_prism.P01, H)
  RETURN square_prism.OB1
}
```

![Diagram of object designs](image)

Figure 3: Example of object design: (a) L-shaped object, (b) object A, (c) object B, (d) prism location

When the user chooses an available modelling operation or constraint, he/she has to define the involved parameters. This can be done through,
an specific value. It is the case of '4' in the previous sequence.

- a symbol that identifies a formal parameter of the model wich will appear on the
heading of the CPSM. In the previous example the parameter 'L'.

- a function or expression that computes and returns the parameter value. It is the
case of the function 'point' that creates a point located by the user on the graphic
screen.

The following options are therefore possible in the previous example:

\[
square\_prism.P01 := \text{Reg\_pol} \ (4, \ 16, \ \text{point}(0.0, \ 0.0, \ 0.0))
\]

where the length of the polygon edge has been fixed to 16.

\[
square\_prism.P01 := \text{Reg\_pol} \ (4, \ 2*\text{L-H}, \ \text{point}(0.0, \ 0.0, \ 0.0))
\]

in this case the number of sides is 4 and the polygon edge size is the evaluation
of \(2 \times L - H\).

The parameter \(\text{square\_prism.P01}\) in the \text{Paral\_sweep} operation indicates that the polygon
to be operated is the result of the \text{Reg\_pol} invocation. However, also it is possible to do:

\[
square\_prism.0B1 := \text{Paral\_sweep} \ (\text{Reg\_pol} \ (4, \ \text{L}, \ \text{point}(0., \ 0., \ 0.),\text{H}))
\]

Now, the user can simply ask for two instances of \text{square\_prism} and define in a graphic
way a number of constraints in order to fix the relative location of both prisms. The final
object is generated by means of a boolean union operation. The constructive parametric
solid model that will be obtained for the final L-shaped object will be,

\[
\text{MODEL L\_object (L,F) return object}
\]

\[
\{
L\_object.0B1 := \text{square\_prism} \ (L, \ 2*\text{L})
\]

\[
L\_object.0B2 := \text{square\_prism} \ (L, \ 2*\text{L+F})
\]

\[
\text{Equal\_normal} \ (L\_object.0B1.F2, \ L\_object.0B2.F1)
\]

\[
\text{Coincident\_2P} \ (L\_object.0B1.\text{PT5}, \ L\_object.0B2.\text{PT1})
\]

\[
\text{Coincident\_2P} \ (L\_object.0B1.\text{PT6}, \ L\_object.0B2.\text{PT2})
\]

\[
L\_object.0B3 := \text{Union} \ (L\_object.0B1, \ L\_object.0B2)
\]

\[
\text{RETURN} \ L\_object.0B3
\]

22
In this second step, three constraints have been introduced. First, the user selects the top faces of both prisms, figures 3-b and 3-c, and asks for coincidence of the normal vectors. Then, coincidence of the point 5 of prism L.object.OB1 with point 1 of prism L.object.OB2 is required, and the same with point 6 of prism L.object.OB1 and point 2 of prism L.object.OB2, figures 3-b and 3-c. The user works by graphically selecting the points to be coincident, and the system generates the corresponding constraint sentences in the model description by using the point ordering in the data structure associated to the square_prism model, figure 3. The set of imposed constraints produces the relative location of the prisms indicated in figure 3-d, while the final union operation generates the model of the parametric object in figure 3-a. It must be observed that the spatial location of the generated object is irrelevant in most cases. Both prisms have been designed in a general location, but the right assembly has been obtained by means of the imposed constraints.

6 The Internal Model Representation and Evaluation

As already stated in the previous section, an editable representation of generic objects (CPSM) is suitable for storage and transmission purposes. However, an specific representation of both geometry and constraints is necessary for the evaluation of specific objects and to support interaction during the design process.

The Internal Model Representation consists of an implicit graph containing points related with constraints that have appeared in the CPMS design process. Nodes of the graph represent points of the model. They initially contain the default geometry of the points. Arcs of the graph represent constraints between points.

Using the object oriented paradigm, points of the Internal Model Representation are objects of the class point. Every object of this class has as attributs the geometry of the point and a list of constraints. Each constraint is implemented by one object of the class constraint. The class point has the following methods: propagation_mechanism, set_geometry, get_geometry, subscribe_constraint. The class constraint has the following attributs: a pointer to the function that solves the constraint, a list of the constraint arguments and a 'ticket'. The 'ticket' value keeps the temporal sequence of the constraints and is used for constraint propagation during the evaluation phase. The methods defined in the class constraint are: Set_pointer_function, Set_arguments, Set_Ticket, Get_Ticket, Function_solve_call.

The Internal Model Representation supports five different types of constraints between points.
**Point-to-point**: These are constraints between points in the same polygon or representative points. Representative points are defined for each polygon and each object in the model.

**Polygon-to-polygon**: Constraints between representative points of different polygons

**Object-to-object**: These are constraints between representative points of different objects.

**Point-in-polygon**: A circular constraint that relates points belonging to a same polygon

**Point-in-object**: Constraint that cyclically relates the points of an object.

When the user creates a four sided regular polygon, the IMR contains the geometry of the four vertices plus the Point-to-point constraints representing edge geometrical constraints plus a loop of implicit point-in-polygon constraints (figure 4).

![Figure 4: Evolution of Internal Model Representation](image)

A := Reg_pol (d/2,4)

![Figure 5: Constraints in Internal Model Representation](image)
When a parallel sweep is performed, the minimum geometry of the extruded object is generated. The IMR contains a copy of the initial polygon plus one Polygon-to-polygon constraint plus a loop of implicit Point-in-object constraint (figure 5). From this point the object created can be related with other objects through Object-to-object constraint.

Given a set of defined parameter values, the CPSM will be evaluated to an specific object (figure 2). The evaluation process includes the following steps:

- The Internal Model Representation is automatically generated from the CPSM.
- The Internal Model Representation is validated in order to detect inconsistencies and overconstraints.
- Finally, a numerical evaluator generates the specific object model from the internal representation and the parameter values. The evaluator first deals with non cyclic constraints, and solves the remaining cycles in a second step.

7 Examples and Discussion

The Constructive Parametric Solid Model of an object is a textual, procedural description containing both modelling operations and associated explicit and implicit constraints. It can be used for modelling either 2D or 3D generic objects. Generic models are implemented as functions that return values that are of a valid type.

In this section we present both the design process and the generation of the constructive parametric solid model of different types of objects based on the object of figure 8. The first step consists on the generation of the parametric polygon indicated in figure 6. The model depends on the parameters: E, D and H. The user directly draws the closed polygon using a graphical user interface, and afterwards indicates several constraints on the vertical dimensions. The resulting constructive model of the parametric polygon includes both the input polygon with the specific coordinates input by the user, and the distance constraints,

MODEL four_prof(D, H, E) return object
{
  four_prof.0B1 := Closed_pol(Point3D(0.,0.,0.), Point3D(10.,0.,0.),
                            Point3D(10.,8.,0.), Point3D(20.,8.,0.),
                            Point3D(20.,10.,0.), Point3D(8.,10.,0.),
                            Point3D(8.,2.,0.), Point3D(0.,2.,0.))
  X_dist(four_prof.PT1, four_prof.PT2, D, DEF_REF())
}
Figure 6: Example of polygon design

```
Coincident_X(four_prof.PT2, four_prof.PT3, DEF_REF())
Coincident_Y(four_prof.PT3, four_prof.PT4, DEF_REF())
Y_dist(four_prof.PT4, four_prof.PT5, E, DEF_REF())
Y_dist(four_prof.PT2, four_prof.PT5, H, DEF_REF())
X_dist(four_prof.PT5, four_prof.PT6, D, DEF_REF())
Coincident_X(four_prof.PT6, four_prof.PT7, DEF_REF())
Coincident_Y(four_prof.PT7, four_prof.PT8, DEF_REF())
Y_dist(four_prof.PT8, four_prof.PT1, E, DEF_REF())
RETURN four_prof.0B1
```

Now, the 3D solid 'four' can be generated by means of a parallel sweep (figure 7). The generated model will be,

```
MODEL four(L) return object
{
    four.PL1 := four_prof(L, 2*L, L/4)
    four.0B1 := Paral_sweep(four.PL1, L/2)
    RETURN four.0B1
}
```
Figure 7: Example of 3D object design

The solid generated could be modified through a parallel sweep operation on one of the object faces (figure 8).
Figure 8: Example of modelling operation on a object designed

MODEL final_four(L) return object
{
    final_four.OB1 := four(L)
    final_four.PL11 := Reg_pol(4, L/4, Centroid(final_four.PL4), XY_REF)
    final_four.OB2 := Mult_sweep(final_four.OB1, final_four.PL4, -L)
    RETURN final_four.OB2
}

Finally, the generation of the object showed in figure 9 is performed through instances of the final_four object just defined. Between objects it is possible to define constraints that fix the pose.
MODEL support(D) return object
{
    support.OB1 := final_four(D)
    support.OB2 := final_four(D)
    Equal_normal(support.OB1.PL1, support.OB2.PL1)
    Coincident_2P(support.OB1.PT1, support.OB2.PT9)
    Coincident_2P(support.OB1.PT9, support.OB2.PT1)
    support.OB3 := Union(support.OB1, support.OB2)
    RETURN support.OB3
}

All intermediate models are stored on a model library, and can be instantiated in different applications. Any model is a non-evaluated representation of the parametric family, and can be evaluated by means of a geometric interpreter, figure 2.

8 Conclusions

A new constraint-based modelling scheme has been proposed, based on the definition of constructive parametric solid models. A non-evaluated procedural model of the parametric family is automatically generated during the design process of a single object of the family. General modelling operations together with geometric constraints can be mixed with no restriction during the design step.

Constructive parametric solid models support instantiating of predefined models, variable
topologies, parametric geometric operations in 1D, 2D and 3D, and operations with structural constraints. A specification of the model definition language has been presented and discussed through several examples.

Future work will involve the analysis of constraints consistency, the generation of specific tools for the detection of overconstraints, efficient mechanisms for deletion and editing of geometric parts, and aspects related to the validity and range of the parameters. On the other hand the set of supported modelling operations and geometric constraints will be extended beyond the kernel that has been presented in the appendix.

9 REFERENCES


[LeA85] K.Lee, G.Andrews. Inference of the Positions of Components in an assembly:


LSI-94-1-R "Logspace and logtime leaf languages", Birgit Jenner, Pierre McKenzie, and Denis Thérien.

LSI-94-2-R "Degrees and reducibilities of easy tally sets", Montserrat Herno.

LSI-94-3-R "Isothetic polyhedra and monotone boolean formulae", Robert Juan-Arinyo.

LSI-94-4-R "Una modelización de la incompletitud en los programas" (written in Spanish), Javier Pérez Campo.


LSI-94-6-R "Construction of the Face Octree model", Núria Pla-Garcia.

LSI-94-7-R "On the expected depth of boolean circuits", Josep Díaz, María J. Serna, Paul Spirakis, and Jacobo Torán.

LSI-94-8-R "A transformation scheme for double recursion", José L. Balcázar.

LSI-94-9-R "On architectures for federated DB systems", Félix Saltor, Benet Campderrich, and Manuel García-Solaco.

LSI-94-10-R "Relative knowledge and belief: SKL preferred model frames", Matías Alvarado.

LSI-94-11-R "A top-down design of a parallel dictionary using skip lists", Joaquim Gabarró, Conrado Martínez, and Xavier Meseguer.

LSI-94-12-R "Analysis of an optimized search algorithm for skip lists", Peter Kirschenhofer, Conrado Martínez, and Helmut Prodinger.

LSI-94-13-R "Bases de dades bitemporals" (written in Catalan), Carme Martín and Jaume Sistac.

LSI-94-14-R "A volume visualization algorithm using a coherent extended weight matrix", Daniela Tost, Anna Puig, and Isabel Navazo.

LSI-94-15-R "Deriving transaction specifications from deductive conceptual models of information systems", María Ribera Sancho and Antoni Olivé.


LSI-94-18-R “Medición del factor modificabilidad en el proyecto LESD” (written in Spanish), Núria Castell and Olga Slavkova.

LSI-94-19-R “Algorismes paral·lels SIMD d’extracció de models de fronteres a partir d’arbres octals no exactes” (written in Catalan), Jaume Solé and Robert Juan-Arinyo.

LSI-94-20-R “Una paral·leització SIMD de la conversió d’objectes codificats segons el model de fronteres al mdel d’octrees clàssics” (written in Catalan), Jaume Solé and Robert Juan-Arinyo.

LSI-94-21-R “Clausal proof nets and discontinuity”, Glyn Morrill.


LSI-94-24-R “On infinite sequences (almost) as easy as π”, José L. Balcázar, Ricard Gavaldà, and Montserrat Hermo.

LSI-94-25-R “Updating knowledge bases while maintaining their consistency”, Ernest Teniente and Antoni Olivé.

LSI-94-26-R “Structural facilitation and structural inhibition”, Glyn Morrill.


LSI-94-29-R “Towards a VRQS representation”, Mariona Taulé Delor.


LSI-94-32-R “LDB/LKB integration”, German Rigau, Horacio Rodríguez, and Jordi Turmo.

LSI-94-33-R “Learning more appropriate selectional restrictions”, Francesc Ribas.

LSI-94-34-R “Oracles and queries that are sufficient for exact learning”, Nader H. Bashouty, Richard Cleve, Ricard Gavaldà, Sampath Kannan, and Christino Tamon.
LSI–94–35–R “Acquisition of lexical translation relations from MRDs”, Ann Copestake, Ted Briscoe, Piek Vossen, Alicia Ageno, Irene Castellón, Francesc Ribas, German Rigau, Horacio Rodríguez, and Anna Samiotou.


Copies of reports can be ordered from:

Nuria Sánchez
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Pau Gargallo, 5
08028 Barcelona, Spain
secretlsi@lsi.upc.es