Design Automation Using Script Languages. High-Level CAD Templates in Non-Parametric Programs

R. Moreno[1], A. M. Bazán[2]

[1] University of Granada, Dpto. Expresión Gráfica, E.T.S. Ingeniería de Edificación, 18071 Granada, Spain
[2] UPM, Dpto. Ingeniería Civil: Construcción, E.T.S. Ingeniería de Caminos, Canales y Puertos, 28040 Madrid, Spain

rmc@ugr.es

Abstract. The main purpose of this work is to study the advantages offered by the application of traditional techniques of technical drawing in processes for automation of the design, with non-parametric CAD programs, provided with scripting languages. Given that an example drawing can be solved with traditional step-by-step detailed procedures, is possible to do the same with CAD applications and to generalize it later, incorporating references. In today's modern CAD applications, there are striking absences of solutions for building engineering: oblique projections (military and cavalier), 3D modelling of complex stairs, roofs, furniture, and so on. The use of geometric references (using variables in script languages) and their incorporation into high-level CAD templates allows the automation of processes. Instead of repeatedly creating similar designs or modifying their data, users should be able to use these templates to generate future variations of the same design. This paper presents the automation process of several complex drawing examples based on CAD script files aided with parametric geometry calculation tools. The proposed method allows us to solve complex geometry designs not currently incorporated in the current CAD applications and to subsequently create other new derivatives without user intervention. Automation in the generation of complex designs not only saves time but also increases the quality of the presentations and reduces the possibility of human errors.

1. Introduction

The aim of the ongoing research is to shift from manual design of disposable geometries to CAD automation by introducing high-level generic geometry templates. Instead of repeatedly modelling similar instances of objects, engineers should be able to create more generic models that can represent entire classes of objects. We present new methods to eliminate non-creative geometric drawing and modelling by performing reuse on specific design features. The goal is to allow engineers to work on a higher abstraction level where the use of low-level CAD functions during the drawing and modelling phase is minimized if not fully eradicated [1].

Descriptive geometry provides insight into the structure and metric properties of spatial objects, processes, and principles. Descriptive geometry courses cover not only projection theory but also modelling techniques for curves, surfaces, and solids, thus offering insight into a broad variety of geometric shapes [2].

CAD software is used to increase designer productivity, improve design quality and communications through documentation, and create databases for manufacturing. As the CAD modelling techniques become more and more advanced, it is necessary to complete product modelling and design changes faster than ever [4]. Updating assemblies that have hundreds of sub-assemblies and parts manually in 3D modelling software is very complicated and time consuming.

Undoubtedly, once a task is fully defined, computers and machines are unparalleled in executing it repeatedly with great speed and sustained accuracy. To this end, Hopgood [3] states: "computers have therefore been able to remove the tedium from many tasks that were previously performed manually". The process referred to is also called Design Automation (DA) by various researchers. The key phrase here is that many manual tasks have been removed through DA and a natural question would be: why not remove the tedium from all manual tasks? [4].

One of the great benefits of using CAD to create technical drawings is the ability to adapt to suit our company's processes. If we can establish a technical drawing process that we perform frequently, it can be automated. If we've ever had to do the same thing with CAD twice, think about how we could automate it so we never have to do it again.

One of the easiest ways to automate a CAD process is to write a script [5]. In computer programming terms, a script is a program that will run with no interaction from the user. In AutoCAD [6], a script file is an ASCII text file that contains a set of command line instructions to follow, just like an actor reading from a script. AutoCAD script files always have the file extension '.scr'.

AutoLISP [7] is the original and most popular programming language for AutoCAD. The reason for its popularity is that it is a natural extension of the program. No additional software needs to be run, and AutoLISP can run commands that Autodesk and other developers offer in the command window.

The LISP code can be entered directly into the command window or loaded using '.lsp' or '.scr' files. Once a LISP program has been loaded, the built-in functions can be executed from the command window. These functions can be executed similarly to CAD commands, but it is the programmer who decides which messages to display. It is possible to use LISP code with a command macro that is activated from the CAD user interface or from a tool on a palette.

Visual languages can be very useful for helping architecture students understand general programming concepts, but scripting languages are fundamental for implementing generative design systems [8].

It is possible to learn to draw with AutoCAD and to program with AutoLISP for AutoCAD using the manuals and online aids offered by both Autodesk (knowledge.autodesk.com) and other independent developer websites (lee-mac.com, afralisp.net, or cadtutor.com). Self-learning through tutorials and videos is very widespread and numerous websites are available to solve any questions we may raise using advanced search engines if we search for the terms 'AutoCAD' or 'AutoLisp' as appropriate.

In the design of complex engineering products it is essential to handle cross-couplings and synergies between subsystems [9]. An emerging technique that has the potential to considerably improve the design process is multidisciplinary design optimization (MDO) [1].

MDO requires a concurrent and parametric design framework. Powerful tools in the quest for such frameworks are DA and knowledge-based engineering [9]. The required knowledge is captured and stored as rules and facts that will finally be activated upon demand. A crucial challenge is what kind of knowledge to store in order to create generic DA structures and how to store it.

In an effort to address the above challenges, this paper proposes the creation of High-Level CAD templates (HLCts) for the manipulation of geometry and High-Level Analysis templates (HLAts) for concept evaluations.

2. High-level CAD templates in non-parametric CAD

AutoCAD and other compatible applications automatically create object identifiers, hidden during the drawing process, that a user usually does not know. These are necessary for the internal manipulation of the objects, but because of their extensions they are difficult to use learning descriptive geometry procedures.

Through LISP variables it is possible to create geometric and object references similar to those traditionally used in descriptive geometry. These references will be very useful in the detailed description of graphic procedures. In the command window, macros, and script files, the use of CAD drawing commands can be combined with LISP commands, functions, and variables (used as geometric and object references). They are equivalent:

```
_vpoint _r !alfa !beta                 ; in script files and in the command window

(command "_vpoint " "_r" alfa beta)    ; in LISP files and in script files
```

Notes: The comment lines are preceded by semicolons and serve to facilitate the understanding of the code. All AutoCAD commands preceded by the underscore will be executed even if the program is installed in another language. The references used in the script files or in the command window are preceded by the exclamation mark. The keyword "_rotate" and its abbreviation "_r" do not use the quotation marks in the commands but use them in the command function. LISP functions (in parentheses) can be used as arguments in CAD commands. The commands can be used in LISP using the command function.

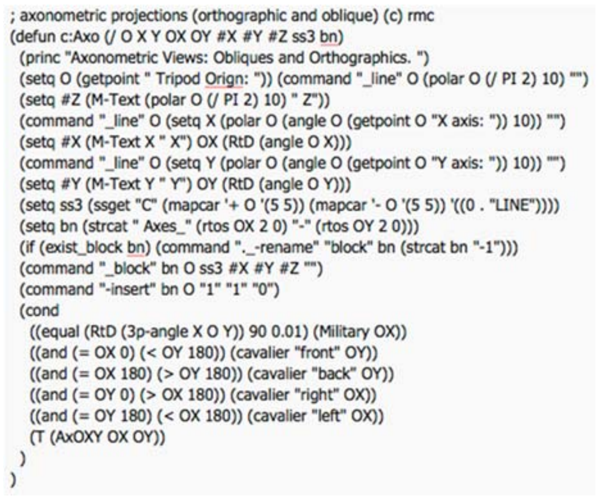
Addressing a simple design task using advanced CAD applications is not difficult. But some specific designs require the application of traditional methods of descriptive geometry. To solve them efficiently requires a detailed analysis of possible combinations. The incorporation of mathematical functions with geometrical and topological calculations is fundamental to avoid drawing step by step. This is the proposed case of HLCts with non-parametric CAD using script languages.

Procedure for creating HLCts is as follows: Examples are selected from standard tasks where CAD applications do not offer solutions. It can be solved first as paper-and-pencil sketches and later with AutoCAD. The command history is extracted and summarized it in script files. To avoid unexpected errors when executing drawing commands using scripts, it is essential to disable certain drawing aids, visible in the status line, and to activate them when finished. Reference variables are created with the parametric data and the objects are drawn. We calculate the derived references that can be reused and redraw. A later analysis will allow us to process the information in a global way using variables, functions, and commands created with AutoLISP. A detailed analysis and design allows us to anticipate possible strategies to complete the design. We create the analysis functions necessary to deal with any situation. Finally, we carry out debugging to remove possible errors.

We present and discuss the results of two practical examples of the workplace: 1) *automation of axonometries* and 2) *automation of ellipses with conjugated diameters*. The first example uses the bearing of the coordinate axes to determine the type of projection. The second example uses the Mannheim method.

3. Results and discussion

In the automation of axonometries, we calculate the viewing direction in space from the tripod of axes on the paper and we obtain blocks with the projections.



```
; axonometric projections (orthographic and oblique) (c) rmc
(defun c:Axo (/ O X Y OX OY #X #Y #Z ss3 bn)
  (princ "Axonometric Views: Obliques and Orthographics. ")
  (setq O (getpoint " Tripod Orign: ")) (command "_line" O (polar O (/ PI 2) 10) "")
  (setq #Z (M-Text (polar O (/ PI 2) 10) " Z"))
  (command "_line" O (setq X (polar O (angle O (getpoint O "X axis: ")) 10)) "")
  (setq #X (M-Text X " X") OX (RtD (angle O X)))
  (command "_line" O (setq Y (polar O (angle O (getpoint O "Y axis: ")) 10)) "")
  (setq #Y (M-Text Y " Y") OY (RtD (angle O Y)))
  (setq ss3 (ssget "C" (mapcar '+ O '(5 5)) (mapcar '- O '(5 5)) '((0 . "LINE"))))
  (setq bn (strcat " Axes_" (rtos OX 2 0) "-" (rtos OY 2 0)))
  (if (exist_block bn) (command "._-rename" "block" bn (strcat bn "-1")))
  (command "_block" bn O ss3 #X #Y #Z "")
  (command "-insert" bn O "1" "1" "0")
  (cond
   ((equal (RtD (3p-angle X O Y)) 90 0.01) (Military OX))
   ((and (= OX 0) (< OY 180)) (cavalier "front" OY))
   ((and (= OX 180) (> OY 180)) (cavalier "back" OY))
   ((and (= OY 0) (> OX 180)) (cavalier "right" OX))
   ((and (= OY 180) (< OX 180)) (cavalier "left" OX))
   (T (AxOXY OX OY))
  )
)
```

Figure 1. LISP – Axo command

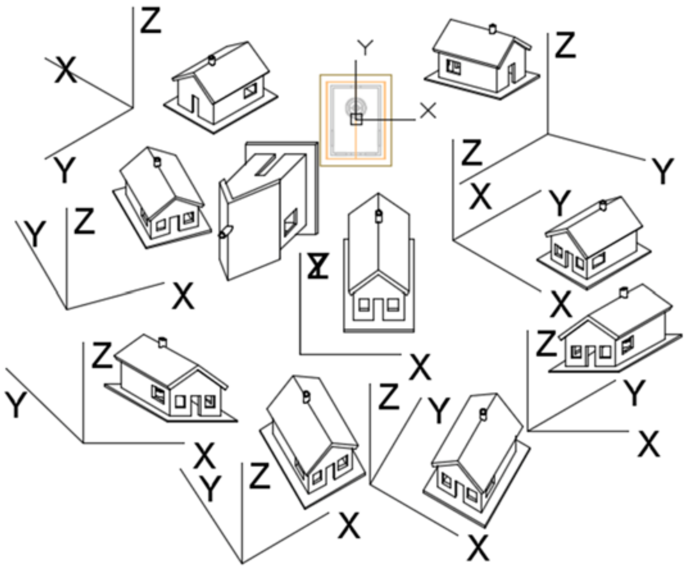
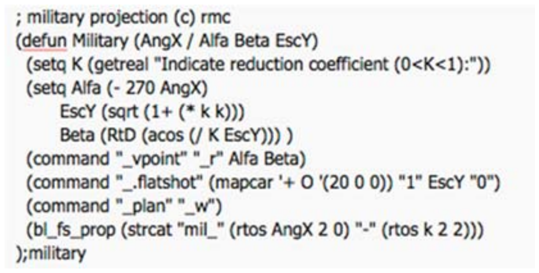


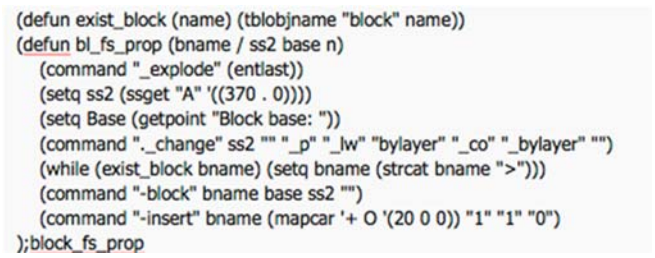
Figure 2. Axonometric projections

In oblique projection, we look for an orthographic intermediate projection, where we calculate the angle *alpha* of the X-axis as a function of the angle of the reduced axis and the angle *beta* from the XY plane as a function of the coefficient *k* of reduction. The block of the intermediate projection is inserted with a scale *EscY* as a function of the coefficient of reduction. The line thickness, base point, and colour errors in the projections are corrected. This method can be applied to obtain shadows on horizontal, frontal, or profile planes.



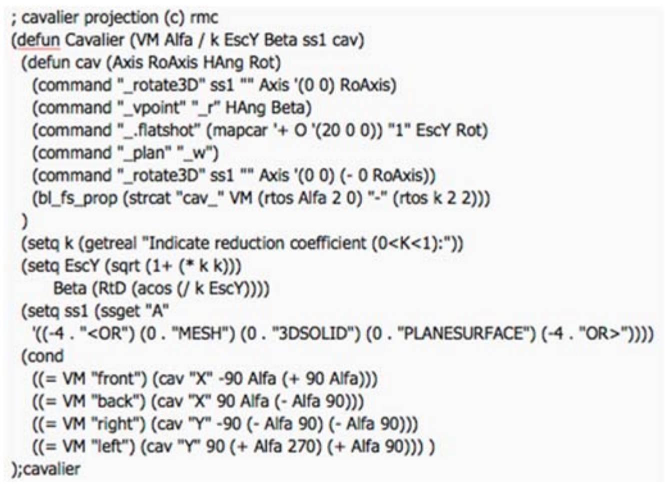
```
; military projection (c) rmc
(defun Military (AngX / Alfa Beta EscY)
  (setq K (getreal "Indicate reduction coefficient (0<K<1):"))
  (setq Alfa (- 270 AngX)
        EscY (sqrt (1+ (* k k)))
        Beta (RtD (acos (/ K EscY))) )
  (command "_vpoint" "_r" Alfa Beta)
  (command "_.flatshot" (mapcar '+ O '(20 0 0)) "1" EscY "0")
  (command "_plan" "_w")
  (bl_fs_prop (strcat "mil_" (rtos AngX 2 0) "-" (rtos k 2 2)))
);military
```

Figure 3. LISP – Military function



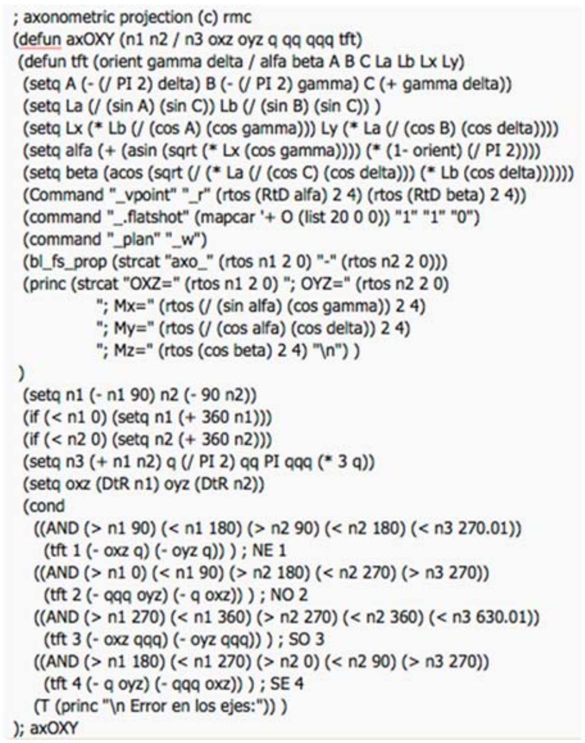
```
(defun exist_block (name) (tblobjname "block" name))
(defun bl_fs_prop (bname / ss2 base n)
    (command "_explode" (entlast))
    (setq ss2 (ssget "A" '((370 . 0))))
    (setq Base (getpoint "Block base: "))
    (command "._change" ss2 "" "_p" "_lw" "bylayer" "_co" "_bylayer" "")
    (while (exist_block bname) (setq bname (strcat bname ">")))
    (command "-block" bname base ss2 "")
    (command "-insert" bname (mapcar '+ O '(20 0 0)) "1" "1" "0")
);block_fs_prop
```

Figure 4. LISP – Flat shot block properties function



```
; cavalier projection (c) rmc
(defun Cavalier (VM Alfa / k EscY Beta ss1 cav)
  (defun cav (Axis RoAxis HAng Rot)
    (command "_rotate3D" ss1 "" Axis '(0 0) RoAxis)
    (command "_vpoint" "_r" HAng Beta)
    (command "_.flatshot" (mapcar '+ O '(20 0 0)) "1" EscY Rot)
    (command "_plan" "_w")
    (command "_rotate3D" ss1 "" Axis '(0 0) (- 0 RoAxis))
    (bl_fs_prop (strcat "cav_" VM (rtos Alfa 2 0) "-" (rtos k 2 2)))
  )
  (setq k (getreal "Indicate reduction coefficient (0<K<1):"))
  (setq EscY (sqrt (1+ (* k k)))
        Beta (RtD (acos (/ k EscY))))
  (setq ss1 (ssget "A"
    '((-4 . "<OR") (0 . "MESH") (0 . "3DSOLID") (0 . "PLANESURFACE") (-4 . "OR>"))))
  (cond
    ((= VM "front") (cav "X" -90 Alfa (+ 90 Alfa)))
    ((= VM "back") (cav "X" 90 Alfa (- Alfa 90)))
    ((= VM "right") (cav "Y" -90 (- Alfa 90) (- Alfa 90)))
    ((= VM "left") (cav "Y" 90 (+ Alfa 270) (+ Alfa 90))) )
);cavalier
```

Figure 5. LISP – Cavalier function

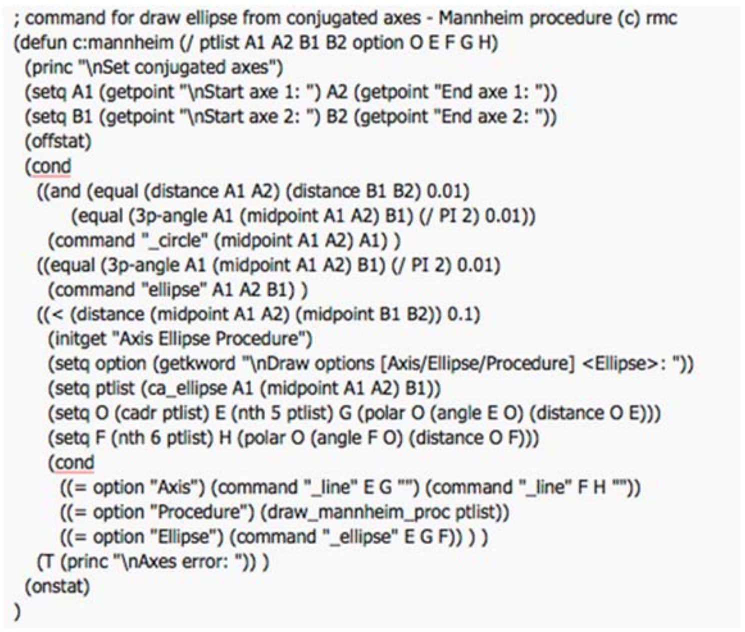


```
; axonometric projection (c) rmc
(defun axOXY (n1 n2 / n3 oxz oyz q qq qqq tft)
 (defun tft (orient gamma delta / alfa beta A B C La Lb Lx Ly)
  (setq A (- (/ PI 2) delta) B (- (/ PI 2) gamma) C (+ gamma delta))
  (setq La (/ (sin A) (sin C)) Lb (/ (sin B) (sin C)) )
  (setq Lx (* Lb (/ (cos A) (cos gamma))) Ly (* La (/ (cos B) (cos delta))))
  (setq alfa (+ (asin (sqrt (* Lx (cos gamma)))) (* (1- orient) (/ PI 2))))
  (setq beta (acos (sqrt (/ (* La (/ (cos C) (cos delta))) (* Lb (cos delta))))))
  (Command "_vpoint" "_r" (rtos (RtD alfa) 2 4) (rtos (RtD beta) 2 4))
  (command "_.flatshot" (mapcar '+ O (list 20 0 0)) "1" "1" "0")
  (command "_plan" "_w")
  (bl_fs_prop (strcat "axo_" (rtos n1 2 0) "-" (rtos n2 2 0)))
  (princ (strcat "OXZ=" (rtos n1 2 0) "; OYZ=" (rtos n2 2 0)
              "; Mx=" (rtos (/ (sin alfa) (cos gamma)) 2 4)
              "; My=" (rtos (/ (cos alfa) (cos delta)) 2 4)
              "; Mz=" (rtos (cos beta) 2 4) "\n") )
 )
  (setq n1 (- n1 90) n2 (- 90 n2))
  (if (< n1 0) (setq n1 (+ 360 n1)))
  (if (< n2 0) (setq n2 (+ 360 n2)))
  (setq n3 (+ n1 n2) q (/ PI 2) qq PI qqq (* 3 q))
  (setq oxz (DtR n1) oyz (DtR n2))
  (cond
   ((AND (> n1 90) (< n1 180) (> n2 90) (< n2 180) (< n3 270.01))
    (tft 1 (- oxz q) (- oyz q)) ) ; NE 1
   ((AND (> n1 0) (< n1 90) (> n2 180) (< n2 270) (> n3 270))
    (tft 2 (- qqq oyz) (- q oxz)) ) ; NO 2
   ((AND (> n1 270) (< n1 360) (> n2 270) (< n2 360) (< n3 630.01))
    (tft 3 (- oxz qqq) (- oyz qqq)) ) ; SO 3
   ((AND (> n1 180) (< n1 270) (> n2 0) (< n2 90) (> n3 270))
    (tft 4 (- q oyz) (- qqq oxz)) ) ; SE 4
   (T (princ "\n Error en los ejes:")) )
); axOXY
```

Figure 6. Lisp – Orthographic axonometric function

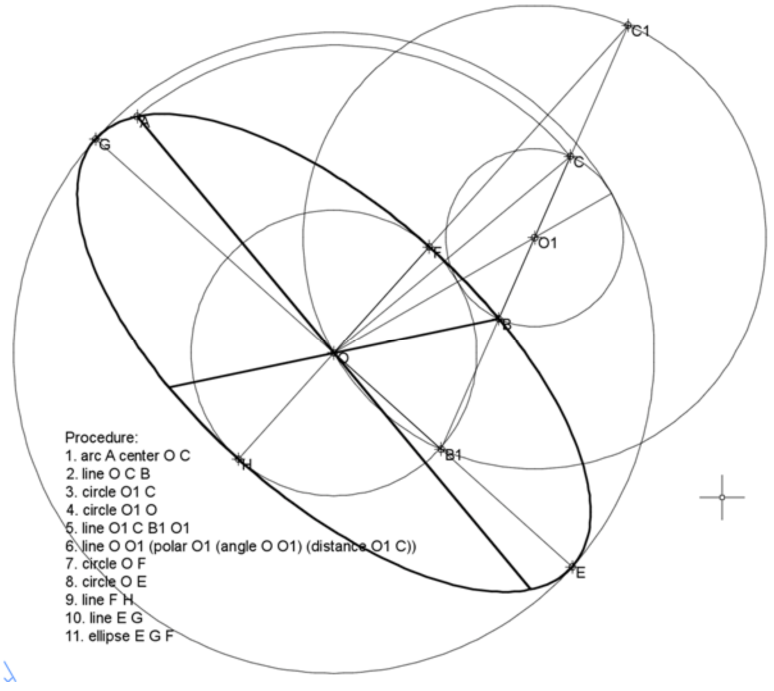
In the orthographic projection, we use the graphic method of the fundamental triangle or of traces for the calculation of the angles of the X-axis and XY-plane.

In the *automation of ellipses with conjugate diameters*, we follow the Mannheim method for obtaining the principal axes, ellipse, or procedure step-by-step (as a learning tool). This method can be applied to obtain projections of revolving surfaces: cone, cylinder, sphere and toroid.



```
; command for draw ellipse from conjugated axes - Mannheim procedure (c) rmc
(defun c:mannheim (/ ptlist A1 A2 B1 B2 option O E F G H)
 (princ "\nSet conjugated axes")
 (setq A1 (getpoint "\nStart axe 1: ") A2 (getpoint "End axe 1: "))
 (setq B1 (getpoint "\nStart axe 2: ") B2 (getpoint "End axe 2: "))
 (offstat)
 (cond
  ((and (equal (distance A1 A2) (distance B1 B2) 0.01)
       (equal (3p-angle A1 (midpoint A1 A2) B1) (/ PI 2) 0.01))
   (command "_circle" (midpoint A1 A2) A1) )
  ((equal (3p-angle A1 (midpoint A1 A2) B1) (/ PI 2) 0.01)
   (command "ellipse" A1 A2 B1) )
  ((< (distance (midpoint A1 A2) (midpoint B1 B2)) 0.1)
   (initget "Axis Ellipse Procedure")
   (setq option (getkword "\nDraw options [Axis/Ellipse/Procedure] <Ellipse>: "))
   (setq ptlist (ca_ellipse A1 (midpoint A1 A2) B1))
   (setq O (cadr ptlist) E (nth 5 ptlist) G (polar O (angle E O) (distance O E)))
   (setq F (nth 6 ptlist) H (polar O (angle F O) (distance O F)))
   (cond
    ((= option "Axis") (command "_line" E G "") (command "_line" F H ""))
    ((= option "Procedure") (draw_mannheim_proc ptlist))
    ((= option "Ellipse") (command "_ellipse" E G F)) ) )
  (T (princ "\nAxes error: ")) )
 (onstat)
)
```

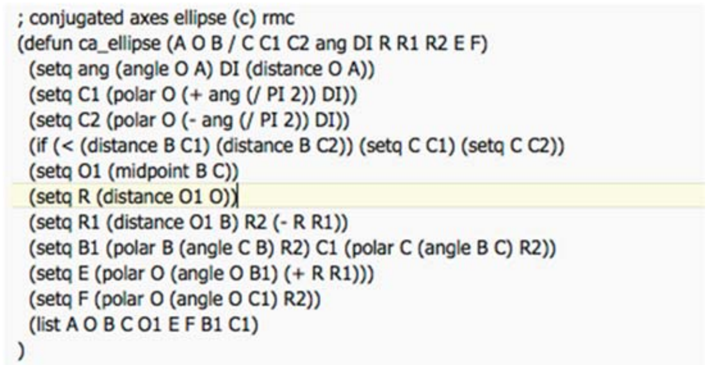
Figure 7. Lisp – Mannheim command



Procedure:
1. arc A center O C
2. line O C B
3. circle O1 C
4. circle O1 O
5. line O1 C B1 O1
6. line O O1 (polar O1 (angle O O1) (distance O1 C))
7. circle O F
8. circle O E
9. line F H
10. line E G
11. ellipse E G F

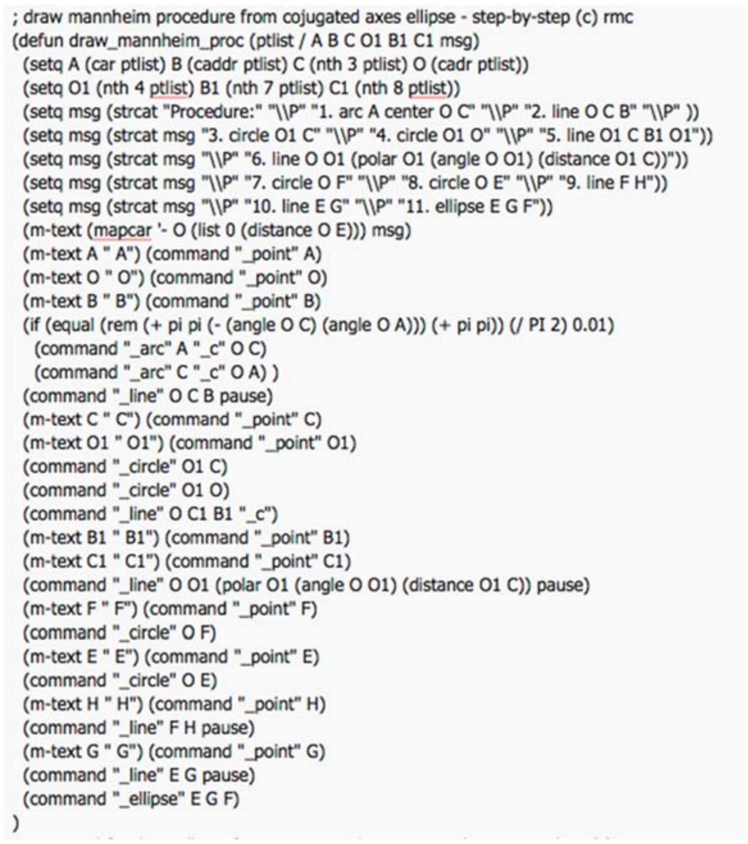
Figure 8. Mannheim procedure

The exposed typology of HLCts of geometry and modeling, based on ontologies, represents a great advance for the automatic generation of designs, which can be repeated with multiple parametric data. But only a concurrent and parametric design structure can determine what kind of knowledge it is necessary to store to discover generic structures of design automation and how to store it.



```
; conjugated axes ellipse (c) rmc
(defun ca_ellipse (A O B / C C1 C2 ang DI R R1 R2 E F)
  (setq ang (angle O A) DI (distance O A))
  (setq C1 (polar O (+ ang (/ PI 2)) DI))
  (setq C2 (polar O (- ang (/ PI 2)) DI))
  (if (< (distance B C1) (distance B C2)) (setq C C1) (setq C C2))
  (setq O1 (midpoint B C))
  (setq R (distance O1 O))
  (setq R1 (distance O1 B) R2 (- R R1))
  (setq B1 (polar B (angle C B) R2) C1 (polar C (angle B C) R2))
  (setq E (polar O (angle O B1) (+ R R1)))
  (setq F (polar O (angle O C1) R2))
  (list A O B C O1 E F B1 C1)
)
```

Figure 9. Conjugate diameters ellipse function



```
; draw mannheim procedure from cojugated axes ellipse - step-by-step (c) rmc
(defun draw_mannheim_proc (ptlist / A B C O1 B1 C1 msg)
  (setq A (car ptlist) B (caddr ptlist) C (nth 3 ptlist) O (cadr ptlist))
  (setq O1 (nth 4 ptlist) B1 (nth 7 ptlist) C1 (nth 8 ptlist))
  (setq msg (strcat "Procedure:" "\\P" "1. arc A center O C" "\\P" "2. line O C B" "\\P" ))
  (setq msg (strcat msg "3. circle O1 C" "\\P" "4. circle O1 O" "\\P" "5. line O1 C B1 O1"))
  (setq msg (strcat msg "\\P" "6. line O O1 (polar O1 (angle O O1) (distance O1 C))"))
  (setq msg (strcat msg "\\P" "7. circle O F" "\\P" "8. circle O E" "\\P" "9. line F H"))
  (setq msg (strcat msg "\\P" "10. line E G" "\\P" "11. ellipse E G F"))
  (m-text (mapcar '- O (list 0 (distance O E))) msg)
  (m-text A " A") (command "_point" A)
  (m-text O " O") (command "_point" O)
  (m-text B " B") (command "_point" B)
  (if (equal (rem (+ pi pi (- (angle O C) (angle O A))) (+ pi pi)) (/ PI 2) 0.01)
    (command "_arc" A "_c" O C)
    (command "_arc" C "_c" O A) )
  (command "_line" O C B pause)
  (m-text C " C") (command "_point" C)
  (m-text O1 " O1") (command "_point" O1)
  (command "_circle" O1 C)
  (command "_circle" O1 O)
  (command "_line" O C1 B1 "_c")
  (m-text B1 " B1") (command "_point" B1)
  (m-text C1 " C1") (command "_point" C1)
  (command "_line" O O1 (polar O1 (angle O O1) (distance O1 C)) pause)
  (m-text F " F") (command "_point" F)
  (command "_circle" O F)
  (m-text E " E") (command "_point" E)
  (command "_circle" O E)
  (m-text H " H") (command "_point" H)
  (command "_line" F H pause)
  (m-text G " G") (command "_point" G)
  (command "_line" E G pause)
  (command "_ellipse" E G F)
)
```

Figure 10. Lisp – Mannheim procedure function

HLCts of geometry and modelling can be useful for:

- combining traditional and modern methods of design,
- recognition and generation of designs with geometric patterns,

- serving as a platform for learning from existing designs,
- serving as a development platform for the creation of new templates.

Only the synergy of mathematics, descriptive geometry, and CAD will enable designers to advance in the development of new derivative designs in less time and with less effort. In this sense, Stachel [2] writes that "only people with a deep knowledge of descriptive geometry will be able to make extensive use of CAD programs" and also that "the importance of mathematics continues to increase even though computers take charge of the computational work".

Conclusions

If we tackle a design task with traditional procedures, the same can be done with CAD applications. The study and analysis of the tasks undertaken allow us to carry out generalization of HLCts to achieve their automation. A thorough analysis will allow us to use the HLCts in tasks not initially foreseen.

This document proposes a method of automating existing procedures to produce certain designs. The proposed method is analysed with varied data in the two examples presented, allowing the following conclusions to be drawn:

1. by using automation processes, it is possible to undertake designs without effort by the user;
2. automation in the generation of designs not only saves time but also increases the quality of the results and reduces the possibility of human errors;
3. multidisciplinary optimization of design reduces the learning effort and speeds up the acquisition of graphic skills.

Expert users perform the initial creation of HLCts, but their use and modification do not require advanced knowledge.

Acknowledgment(s)
This paper has been developed with the financing of FEDER funds under the projects TIN2013-40658-P and TIN2016-75850-R.

References

[1] Amadori, Kristian, et al. "Flexible and robust CAD models for design automation." *Advanced Engineering Informatics* 26.2, pp. 180–195, 2012.
[2] Stachel, Hellmuth. "The status of today's Descriptive Geometry related education (CAD/CG/DG) in Europe." *Journal of Graphic Science of Japan 41.* Supplement 1, pp. 15–20, 2007.
[3] Hopgood, Adrian A. "Intelligent Systems for Engineers and Scientists", CRC *Press*, 2012.
[4] Siddesh, S., and B. S. Suresh. "Automation of generating CAD models." *Journal of Mechanical Engineering and Automation* 5.3B, pp. 55–58, 2015.
[5] Ambrosius, Lee. *AutoCAD Platform Customization: User İnterface, AutoLISP, VBA, and Beyond*, *John Wiley and Sons*, 2015.
[6] AutoCAD [computer program]. Autodesk, 2017.
[7] AutoLISP (Part of AutoCAD) [computer program]. Autodesk, 2017.
[8] Celani, Gabriela, and Carlos Eduardo Verzola Vaz. "CAD scripting and visual programming languages for implementing computational design concepts: A comparison from a pedagogical point of view." *International Journal of Architectural Computing* 10.1, pp. 121–137, 2012.
[9] Tarkian, Mehdi. *Design Automation for Multidisciplinary Optimization: A High Level CAD Template Approach*, 2012.