TRIBELON
RIVISTA DI DISEGNO
UNIVERSITÀ DEGLI
STUDI DI FIRENZE

VOL. 1 | N. 2 | 2024
DISEGNO: SPAZI DI INTERAZIONE
DRAWING: SPACES OF INTERACTION

Citation: G. Anzani, *Le estensioni Autocad ActiveX Automation in Visual LISP*, in *Codici grafici*, *TRIBELON*, I, 2024, 2, pp. 114-121.
ISSN (stampa): 3035-143X
ISSN (online): 3035-1421
doi: https://doi.org/10.36253/tribelon-3187
Published: December, 2024

Data Availability Statement: All relevant data are within the paper and its Supporting Information files.
Competing Interests: The Author(s) declare(s) no conflict of interest.
Journal Website: *riviste.fupress.net/tribelon*

CODICI GRAFICI

LE ESTENSIONI AUTOCAD ACTIVEX AUTOMATION IN VISUAL LISP

GIOVANNI ANZANI
University of Florence
giovanni.anzani@unifi.it

In questo secondo numero della rubrica codici grafici, si danno le basi delle estensioni ActiveX[1] *applicandole in ambiente AutoCAD congiuntamente a Visual LISP evoluzione di AutoLISP; le estensioni AutoCAD ActiveX Automation sono un protocollo di comunicazione con cui è possibile manipolare AutoCAD a livello di programmazione, sia dall'interno che dall'esterno di AutoCAD. Ciò avviene esponendo gli oggetti AutoCAD al "mondo esterno" rendendoli accessibili da molti linguaggi e ambienti di programmazione e da altre applicazioni.*
I controlli ActiveX vengono applicati a svariati componenti organizzati secondo una gerarchia, tutti identificabili come oggetti o come collezioni (raccolte di oggetti), entrambi contenitori che a loro volta in quello che viene definito modello a oggetti visualizzabile in fig. 1, rendono disponibili: proprietà che consentono di impostare o restituire informazioni sullo stato di un oggetto; metodi che consentono l'esecuzione di un'azione su un oggetto; eventi che sono azioni avviate dall'utente o occorrenze alle quali un programma risponde.
L'argomento offre lo spunto per la realizzazione di un listato di codice, contenente quarantadue funzioni satellite e due funzioni principali che, date due circonferenze, si occupano di determinare ed opzionalmente disegnare, il loro asse radicale e la circonferenza di Apollonio[2] *a loro associata univocamente.*

ActiveX e AutoLISP

ActiveX include gran parte delle funzionalità AutoLISP e generalmente le sue funzioni sono più veloci e forniscono un accesso più facile alle proprietà degli oggetti rispetto alle corrispondenti funzioni tradizionali AutoLISP.
Quando si lavora con l'interfaccia di programmazione ActiveX in Visual LISP e parzialmente in AutoLISP si lavora con lo stesso modello gerarchico a oggetti, con oggetti, proprietà e metodi ActiveX.
Le applicazioni Visual LISP hanno accesso diretto agli oggetti ActiveX, possono richiamare metodi ActiveX, impostare e recuperare proprietà ActiveX ma non sono in grado di disporre degli eventi ActiveX [3]; le applicazioni ActiveX possono eseguire applicazioni Visual LISP. Per rendere disponibili in autocad le estensioni ActiveX e le correlate funzioni Visual LISP entrambe vanno caricate mandando in esecuzione la funzione **vl-load-com**.
Alcuni tipi di dati non sono condivisibili tra i due ambienti di programmazione ad esempio: la lista deve essere convertita in matrice (un array di tipo variant); per questa ragione, ActiveX e le sue estensioni incorporano in Visual LISP tre nuove tipologie di dati: **Array** (matrici), **Variant** (varianti) e nel modello a oggetti le entità AutoCAD divengono **VLA-OBJECT** (Visual Lisp ActiveX).
Per utilizzare AutoCAD ActiveX Automation efficacemente è necessario avere familiarità con le entità, gli oggetti e le funzionalità di AutoCAD; maggiore è la conoscenza delle proprietà grafiche e non grafiche di un oggetto, più facile sarà manipolarle tramite ActiveX.

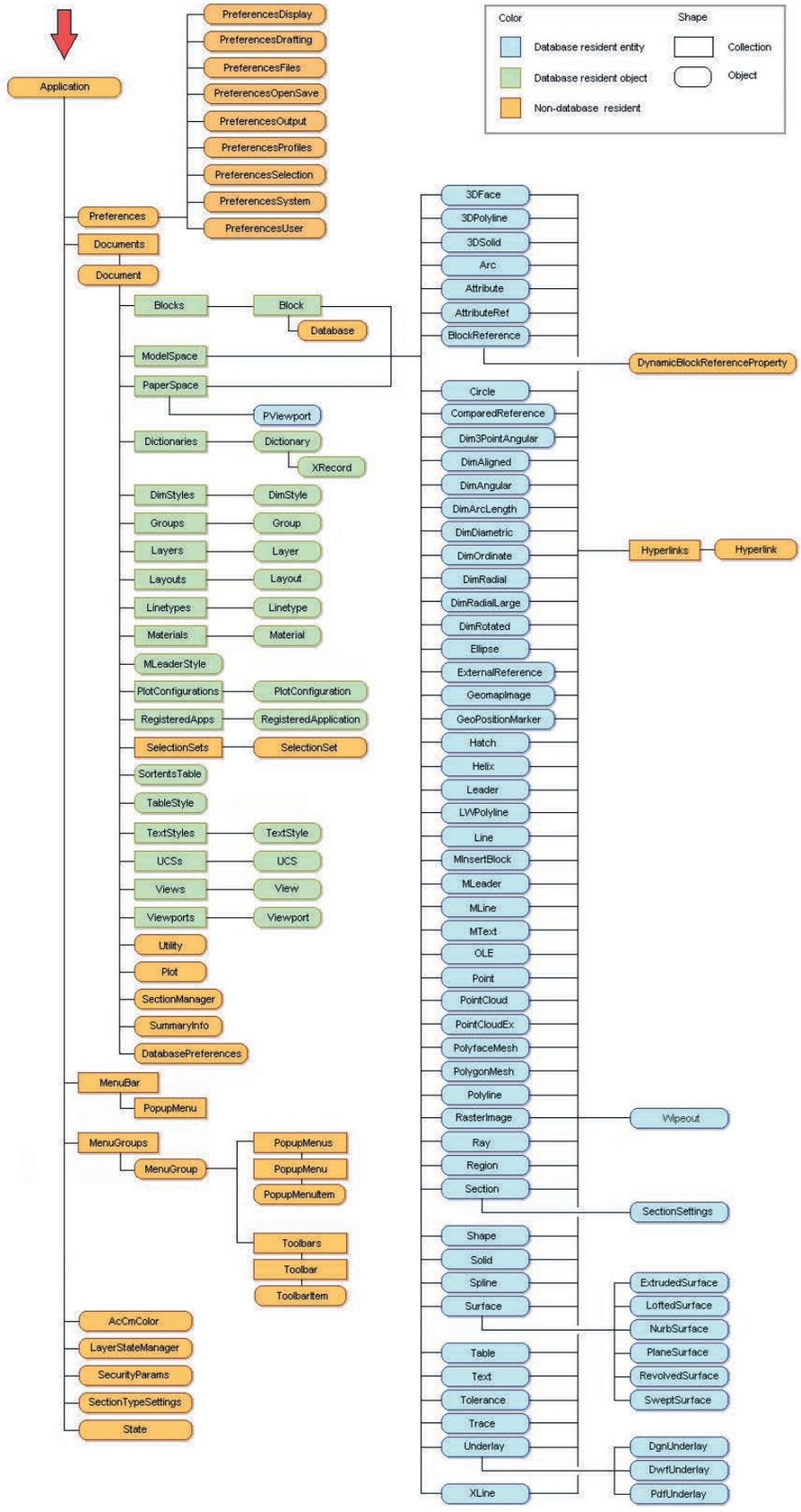
1 **ActiveX** – La tecnologia Microsoft ActiveX Automation è un'estensione che nasce nel 1996 come evoluzione di due precedenti tecnologie Microsoft: OLE (Object Linking and Embedding) e COM (Component Object Model). ActiveX (Active eXtension) è stata pensata per estendere le potenzialità e le funzioni disponibili nelle applicazioni predisposte al suo uso; consente inoltre di semplificare alcuni processi nello sviluppo di software tramite l'implementazione di controlli precodificati in possesso di funzionalità specifiche che vengono poi incorporati e resi disponibili come blocchi di codice nelle applicazioni compatibili.
2 **Circonferenza di Apollonio** – È una curva piana chiusa, luogo geometrico dei punti del piano la cui distanza da due punti fissi detti fuochi ha rapporto costante; i due fuochi risulteranno punti inversi di una inversione circolare avente la circonferenza di Apollonio quale cerchio di inversione, il suo centro quale centro di inversione, ed il quadrato del suo raggio quale potenza dell'inversione. Nel testo (1) in bibliografia sono descritte differenti genesi geometriche della circonferenza di Apollonio.
3 Gli eventi per la loro complessità saranno trattati eventualmente in un altro numero della rubrica.

1 | *Il modello a oggetti AutoCAD ActiveX.*
Gli oggetti sono i principali elementi costitutivi di un'applicazione ActiveX. Ci è generalmente familiare in AutoCAD considerare oggetti le entità grafiche nel disegno, ma nello schema organizzativo di AutoCAD ActiveX, tutti i componenti AutoCAD sono oggetti:
- *Le entità grafiche (linee, archi, testo, quote ...),*
- *Impostazioni di stile (tipi di linea, stili di quota...),*
- *Strutture organizzative (livelli, gruppi, blocchi,...),*
- *Le visualizzazioni del disegno (viste, finestre),*
- *Gli spazi del disegno (modello, carta),*
- *I disegni (intesi come files.dwg aperti)*
- *L'applicazione Autocad*

Gli oggetti sono strutturati in modo gerarchico a partire da un oggetto radice (per AutoCAD l'oggetto Application). La visione di questa struttura gerarchica è chiamata modello a oggetti e mostra quale oggetto fornisce l'accesso al livello successivo di oggetti nella nidificazione strutturata. Ogni oggetto ha un oggetto padre al quale è permanentemente collegato e per il quale è oggetto figlio. Tutti gli oggetti provengono da un singolo oggetto padre radice dal quale è possibile accedere a tutti gli oggetti nell'interfaccia seguendo i collegamenti dalla radice fino agli ultimi oggetti figlio in scala gerarchica nidificata. Nella gerarchia degli oggetti è possibile fare riferimento agli oggetti direttamente o tramite variabili definite dall'utente.
- *(1) – Per uno specifico disegno dall'oggetto radice Application si dovrà entrare nella collezione Documents che contiene tutti i disegni identificando ciascuno di essi come uno specifico oggetto Document.*
- *(2) – Per una specifica circonferenza nello spazio modello di un disegno dall'oggetto radice Application si dovrà entrare nella sua collezione Documents e scegliere lo specifico oggetto Document, da cui si dovrà entrare nella sua collezione ModelSpace e scegliere lo specifico oggetto Circle.*

Nel diagramma si evidenziano delle distinzioni:
- *tra gli oggetti con gli spigoli del rettangolo che ne contiene il nome smussati e le collezioni con gli spigoli del rettangolo che ne contiene il nome privi di smussatura;*
- *tra gli oggetti grafici (le entità grafiche) con il colore di campitura del rettangolo che ne contiene il nome azzurro e gli oggetti non grafici con il colore di campitura del rettangolo che ne contiene il nome arancione o verde;*
- *tra gli oggetti contenuti nel database con il colore di campitura del rettangolo che ne contiene il nome azzurro o verde e gli oggetti non contenuti nel database con il colore di campitura del rettangolo che ne contiene il nome arancione.*

Diagram of the AutoCAD ActiveX Obeject Model



> *La struttura del modello a oggetti ha una gerarchia nidificata: ogni oggetto ha un oggetto padre al quale è permanentemente collegato come oggetto figlio.*

Proprietà e metodi di un oggetto

A ciascun oggetto sono associate proprietà e metodi: le proprietà sono dei dati descrivono aspetti, caratteristiche o stati del singolo oggetto, mentre i metodi sono procedure o funzioni che compiono azioni sul singolo oggetto restituendo il valore di tale azione. Una volta creato un oggetto, è possibile interrogarlo e modificarlo tramite le sue proprietà e metodi.
In termini grammaticali possiamo pensare agli oggetti come a dei nomi, alle proprietà come a degli aggettivi ed ai metodi come a dei verbi.
Oggetti differenti possono avere associate proprietà e metodi analoghi, ad esempio: gli oggetti Circle ed Arc condividono diverse proprietà, tra cui Center, Radius, Layer e diversi metodi tra cui Copy, Offset, Mirror.

Il modello a oggetti in Visual LISP

Visual Lisp fornisce varie specifiche funzioni per leggere e aggiornare le **proprietà** ActiveX (se non di sola lettura), contraddistinte le prime dal prefisso **vla-get**[4] mentre le seconde dal prefisso **vla-put**[5].
Visual Lisp fornisce varie specifiche funzioni per evocare **metodi** ActiveX contraddistinte dal prefisso **vla-**[6].
Visual Lisp aggiunge anche varie funzioni in grado di comunicare con ActiveX contraddistinte dal prefisso **vlax-**[7] si tratta di funzioni più generali applicabili alla lettura/scrittura di oggetti e proprietà ed all'evocazione di metodi.

Confronto AutoLISP Visual LISP

Immaginando di voler accedere al raggio di un cerchio, un confronto tra funzioni standard di AutoLISP e funzioni orientate agli oggetti di Visual LISP risulterà forse utile.
Con le funzioni standard di AutoLISP è necessario utilizzare **entsel** per selezionare l'entità cerchio e salvarne il nome (variabile cir), **entget** per estrarne l'elenco completo di tutte le sue specifiche associazioni codici valori tra cui trovare la proprietà desiderata e salvarlo (variabile ele_cir), e noto il numero di codice (valore di gruppo DXF) associato a quella proprietà (40 per il raggio), ottenerne il valore del raggio con la combinazione delle funzioni cdr e assoc quindi salvarlo (variabile rag):

- (setq cir (car (entsel "Select circle:")))
- (setq ele_cir (entget cir))
- (setq rag (cdr (assoc 40 ele_cir)))

Con le funzioni ActiveX è necessario, prima di qualsiasi altra azione, evocare le funzioni di utilità **vla-Get-Utility** nel contesto del documento attivo **vla-Get-ActiveDocument** di AutoCAD **vlax-Get-Acad-Object**, questo per collocarsi nel contesto desiderato nella gerarchia del modello a oggetti:

- (setq AObj (vlax-get-acad-object))
- (setq AcDo (vla-get-ActiveDocument AObj))
- (setq Ut (vla-get-Utility AcDo))

Sarà quindi possibile utilizzare **vla-GetEntity** per selezionare il cerchio e, noto il nome associato alla proprietà desiderata, ottenerla con la specifica funzione dedicata **vla-Get-Radius**:

- (vla-GetEntity Ut 'cir 'PT "Select circle:")
- (setq rag (vla-get-Radius c ir))

4 **vla-get** – (vla-get-proprietà oggetto) – Recupera il valore di una proprietà da un oggetto:
- (setq lay (vla-get-Layer circle))
- (setq rad (vla-get-Radius circle))

Le espressioni precedenti recuperano i valori del layer di appartenenza e del raggio dall'entità circle e li salvano nelle variabili lay e rad.
Riferendosi alla didascalia della fig. 2 si veda in [6] la funzione GE_PR – (GE_PR # e) predisposta per recuperare una serie di proprietà secondo una lista di indice # da un'entità e, con essa le precedenti operazioni di recupero dall'entità circle saranno:
- (setq lay (GE_PR 2 circle))
- (setq rad (GE_PR 5 circle))

5 **vla-put** – (vla-put-proprietà oggetto valore) – Aggiorna il valore di una proprietà in un oggetto:
- (vla-put-Layer circle "Layer 1")
- (vla-put-Radius circle 1.0)

Le espressioni precedenti aggiornano i valori del layer di appartenenza e del raggio dall'entità circle ai valori "Layer 1" e 1.0.

6 **vla-** – (vla-metodo [var ...]) – Evoca un metodo passandogli variabili in quantità necessaria:
- (setq C1 (vlax-3d-point 1 1 1))
- (setq C2 (vlax-3d-point 2 2 2))
- (setq r1 5.0)
- (setq circle1 (vla-AddCircle modelSpace C1 r1))
- (setq circle2 (vla-Copy circle1))
- (vla-Move circle2 C1 C2)
- (vla-Delete circle1)

Nelle righe di codice precedenti, definiti nelle prime tre righe i punti C1 C2 nel formato vlax-3d-point ed il raggio r1 nella quarta riga si evoca il metodo AddCircle per creare una circonferenza salvata in circle1, nella quinta riga si evoca il metodo Copy per copiare circle1 in una circonferenza circle2, nella sesta riga si evoca il metodo Move per spostare circle2 da C1 a C2 ed infine nella settima riga si evoca il metodo Delete per eliminare circle1.

7 **vlax-** – (vlax-.... [var ...]) – Evoca un metodo passandogli le variabili in quantità necessaria per esso; le funzioni principali di riferimento per la gestione di oggetti proprietà e metodi sono:
- (vlax-get-object prog-id)
- (vlax-create-object prog-id)
- (vlax-get-property obj pro)
- (vlax-put-property obj pro arg)
- (vlax-invoke-method obj met arg [arg ...]).

> *Dall'oggetto radice Application è possibile accedere a cascata ad uno qualsiasi degli altri oggetti oppure alle proprietà o ai metodi assegnati a qualsiasi altro oggetto.*

Con AutoLISP dobbiamo quindi estrarre dall'**entità** Circle la lista completa di tutte le coppie puntate codice valore e quindi estrarne il raggio noto il suo codice; potremo però in linea teorica passare di funzione in funzione i risultati inanellando un'unica riga di codice con un unico settaggio diretto invece di procedere tramite variabili successive:

- (setq rag (cdr (assoc 40 (entget (car (entsel "Select circle:")))))))

Con Visual LISP potremo estrarre dall'oggetto Circle la singola proprietà raggio direttamente noto il suo nome; non potremmo però procedere inanellando i passaggi: nel caso in esame la funzione vla-GetEntity contiene al suo interno il settaggio dell'oggetto Circle ('cir) e non restituisce valori in uscita; sarebbe quindi errato scrivere:

- (setq rag (vla-get-Radius (vla-GetEntity Ut 'cir 'PT "Select circle:")))

Si può comunque in molti contesti Visual LISP procedere col metodo diretto come alternativa al metodo per variabili successive. Ad esempio, per collocarsi in una determinata posizione del modello a oggetti, sarebbe quindi corretto scrivere:

- (vla-GetEntity (vla-get-Utility (vla-get-ActiveDocument (vlax-get-acad-object)))) 'cir 'PT "Select circle:")

- (setq rag (vla-get-Radius cir))

Array

L'array è una struttura dati di forma matriciale di dimensione e tipologia di dato prefissate in fase di creazione. È un oggetto nella cui intestazione sono memorizzate le informazioni sulla sua struttura: il numero di dimensioni che ha (fino a sedici), la lunghezza di ciascuna di tali dimensioni e il tipo univoco di dati che contiene. In termini di allocazione di memoria è un blocco di archiviazione contiguo, i cui elementi sono accessibili tramite indici numerici rendendola particolarmente performante in termini di velocità di accesso.
Gli array implementati utilizzando le estensioni ActiveX sono di tipo safearray una tipologia che non consente l'assegnazione di valori al di fuori dei parametri stabiliti in fase di creazione dell'array sia per posizione che per tipologia del dato.
In Visual LISP, la gestione degli array è risolta da nove funzioni che agiscono come un'interfaccia per richiamare procedure ActiveX e che sono identificate dal prefisso **vlax-** [8].

Variant

Una variante è un contenitore di uso generale in grado di contenere dati dei diversi tipi supportati da ActiveX e che è coinvolto nel trasferimento di informazioni quando vengono utilizzati metodi e proprietà degli oggetti ActiveX di AutoCAD o altre applicazioni Windows gestibili da Visual LISP.
In Visual LISP, la gestione dei dati variant è risolta da quattro funzioni che agiscono come un'interfaccia per richiamare procedure ActiveX e che sono identificate dal prefisso **vlax-** [9].

Collection

AutoCAD raggruppa tutte le istanze di oggetti simili in predefiniti oggetti **Collection**[10] instaurando tra loro una relazione padre/figlio collezione/istanze. Ogni collezione ha un metodo per aggiungere un oggetto al suo interno.

8 **vlax-** – (vlax-.... [var ...]) – Evoca un metodo passandogli le variabili in quantità necessaria per esso; le nove funzioni per la gestione degli array assolvono a quattro finalità: (1) scrittura della struttura, (2) scrittura dei valori contenuti, (3) lettura dei valori contenuti, (4) lettura della struttura:
- vlax-make-safearray (1)
- vlax-safearray-put-element (2)
- vlax-safearray-fill (2)
- vlax-safearray->list (3)
- vlax-safearray-get-element (3)
- vlax-safearray-type (4)
- vlax-safearray-get-dim (4)
- vlax-safearray-get-l-bound (4)
- vlax-safearray-get-u-bound (4)

Un oggetto **array** può contenere i seguenti tipi di dato: Integer, Long, Single, Double, String, Object, Boolean, Variant.
Per creare un array A con A= $\begin{bmatrix} "a" & "b" & "c" \\ "d" & "e" & "f" \end{bmatrix}$
utilizzeremo le due espressioni:
- (setq M (vlax-make-safearray vlax-vbString '(0 . 1) '(0 . 2)))
- (vlax-safearray-fill M '(("a" "b" "c")("d" "e" "f"))

9 **vlax-** – (vlax-.... [var ...]) – Evoca un metodo passandogli le variabili in quantità necessaria per esso; le quattro funzioni per la gestione dei variant assolvono a quattro finalità: (1) scrittura, (2) lettura del tipo di dato, (3) lettura del valore del dato (4) modifica del tipo di dato:
- vlax-make-variant (1)
- vlax-variant-type (2)
- vlax-variant-value (3)
- vlax-variant-change-type (4)

Un oggetto **variant** può contenere i seguenti tipi di dato: Empty, Null, Integer, Long, Single, Double, String, Object, Boolean, Array.
Per creare un variant V (booleano – valori 0/1) utilizzeremo l'espressione:
- (setq V (vlax-make-variant 0 vlax-vbBoolean))

10 **Collection** – Tra la ventina di oggetti Collection esistenti è basilare conoscere almeno:
Documents – Contiene tutti i documenti aperti nella sessione AutoCAD corrente.
ModelSpace – Contiene tutti gli oggetti grafici (entità) presenti nello spazio modello.
PaperSpace – Contiene tutti gli oggetti grafici (entità) presenti nel layout dello spazio carta attivo.
Block Object – Contiene tutte le entità all'interno di una specifica definizione di blocco.
Blocks Collection – Contiene tutti i blocchi nel disegno.

> *Possiamo pensare agli oggetti come a dei nomi (Circle Line ...), alle proprietà come a degli aggettivi (Radius Length ...) ed ai metodi come a dei verbi (Move Rotate ...).*

La maggior parte delle collezioni usa il metodo generale Add ovvero la funzione Visual LISP **vla-Add** [11].
Gli oggetti grafici (entità) vengono solitamente aggiunti alle collezioni utilizzando un metodo derivato dal generale Add e più specializzato Add<EntityName> ovvero la funzione Visual LISP **vla-Add<EntityName>** [12].
Le raccolte di oggetti (Collection), a loro volta oggetti, hanno anche altre proprietà e metodi in comune:

- La proprietà **Count** – (vla-get-Count col) consente il conteggio a partire da zero degli oggetti in una collezione (variabile **col**)

- Il metodo **Item** – (vla-Item col ind) può essere usato per ottenere un qualsiasi oggetto all'interno di una collezione (variabile col) indicizzata (variabile **ind**).

La maggior parte delle raccolte di oggetti sono accessibili tramite l'oggetto **Document** che contiene una proprietà per ciascuno di essi; altre raccolte sono accessibili tramite l'oggetto **Application** che contiene una proprietà per ciascuna di esse.
L'oggetto radice Application contiene l'oggetto raccolta Documents che a sua volta contiene tutti gli oggetti Document (tutti i disegni), ogni oggetto Document (ogni disegno) contiene i suoi specifici oggetti raccolta come ad esempio ModelSpace e PaperSpace che contengono tutti gli oggetti grafici disegnati negli spazi modello o carta.
Dall'oggetto Application è possibile accedere a cascata ad uno qualsiasi degli altri oggetti oppure alle proprietà o ai metodi assegnati a qualsiasi altro oggetto.
L'oggetto Document, che in realtà è un disegno di AutoCAD, si trova nella collezione Documents accessibile tramite la proprietà Documents dell'oggetto Application e fornisce l'accesso a tutti gli oggetti grafici di AutoCAD e alla maggior parte di quelli non grafici.
L'accesso agli oggetti grafici (linee, cerchi, archi e così via) è fornito tramite le collezioni ModelSpace e PaperSpace e l'accesso agli oggetti non grafici (livelli, tipi di linea, stili di testo e così via) è fornito tramite collezioni con nomi simili come ad esempio Layers, Linetypes e TextStyles.

Utility

L'oggetto utilità fornisce funzioni per l'input da parte dell'utente e funzioni di conversione. Le funzioni di input dell'utente sono metodi che richiedono all'utente sulla riga di comando di AutoCAD l'input di vari tipi di dati, come stringhe, numeri interi, numeri reali, punti e così via. Le funzioni di conversione sono metodi che operano su tipi di dati specifici di AutoCAD come punti e angoli, oltre alla gestione di stringhe e numeri.

Un esempio di codice

A causa del poco spazio, non è possibile commentare tutte le 137 righe di codice riportate in fig. 2 e fig. 3, scritte con un'**indentazione** [13] piuttosto compatta, andando a capo il meno possibile bilanciando leggibilità e compattezza del codice. Si descrive dettagliatamente qua solo la procedura [18] e si rimanda alle note di fig. 2 e fig. 3, per una descrizione sommaria di ogni procedura e gruppo di procedure da [0] a [20].

[11] **vla-Add** – Segue una coppia di esempi commentati d'uso del metodo Add: prima di questo definiamo preventivamente il contesto operativo nella gerarchia padre/figlio collocandoci, per i due esempi, nel documento attivo (v. **doc**) di Autocad (v. **cad**):
- (setq cad (vlax-get-acad-object))
- (setq doc (vla-get-ActiveDocument cad))

Come primo esempio aggiungiamo alla collezione **Layers** (v. **Clay**) un oggetto **Layer** (v. **Olay**):
- (setq Clay (vla-get-Layers doc))
- (setq Olay (vla-Add Clay "Layer 1"))

Come secondo esempio definito un punto (v. **PO**) aggiungiamo alla collezione Blocks (v. **CBlon**) un oggetto Block (v. **Oblo**):
- (setq PO (vlax-3d-point 0 0 0))
- (setq CBlon (vla-get-Blocks doc))
- (setq Oblo (vla-Add CBlon PO "Block 1"))

[12] **vla-Add<EntityName>** – Segue una coppia di esempi commentati d'uso del metodo Add<EntityName>: prima di questo definiamo preventivamente il contesto operativo nella gerarchia padre/figlio collocandoci, per i due esempi, nello spazio modello (v. **mos**) del documento attivo (v. **doc**) di Autocad (v. **cad**):
- (setq cad (vlax-get-acad-object))
- (setq doc (vla-get-ActiveDocument cad))
- (setq mos (vla-get-ModelSpace doc))

Come primo esempio, definiti un punto (variabile PO) ed una lunghezza (variabile lun) aggiungiamo un'oggetto Circle (v. **cir**) nello spazio modello del disegno attivo di AutoCAD:
- (setq PO (vlax-3d-point 0 0 0))
- (setq lun 1.0)
- (setq cir (vla-AddCircle mos PO lun))

Come secondo esempio, definito un array vettoriale di coordinate xyz di tre punti (v. **POn**) aggiungiamo un oggetto Polyline (v. **pol**) nello spazio modello del disegno attivo di AutoCAD:
- (setq POn (vlax-make-safearray vlax-vbDouble '(0 . 8)))
- (vlax-safearray-fill POn '(1 1 0 2 2 0 3 3 0))
- (setq pol (vla-AddPolyline mos POn))

La lista di tre punti è definita come safearray con la funzione vlax-make-safearray di dimensione unitaria (matrice vettore) con 9 elementi (x y z x y z x y z) di indici da 0 a 8 popolata con la funzione vlax-safearray-fill di valori di tipo Double (v. **vlax-vbDouble**).

[13] **Indentazione** – È una convenzione utilizzata nella programmazione informatica, per esprimere al meglio la struttura di un codice sorgente. Effettuando nell'editor IDE Visual LISP di AutoCAD un'indentazione automatica standard del codice d'esempio scritto, esso risulta strutturato in 299 righe di codice.

2 | *Le routines proposte (prima parte).*

[0] Nel caricare il codice viene automativamente eseguita la funzione (vl-load-com) operazione indispensabile per rendere disponibili le estensioni AutoCAD ActiveX e le correlate funzioni Visual LISP.

[1] Funzioni dedicate a predispore dei riferimenti alla struttura gerarchia del modello a oggetti riconoscibili dal prefisso MO_ nelle quali una serie di suffissi specificano i riferimenti: al disegno corrente AcDo, allo spazio modello MoSp, alle utility Ut.

[2] Funzioni dedicate alle entità riconoscibili dal prefisso EN_ nelle quali una serie di suffissi specificano lo scopo della funzione: aggiornamento U (Update), eliminazione D (Delete), reperimento dell'entità più recente L (Last). Ad esempio:

- *(EN_LD# 3) elimina le ultime tre entità create.*

[3] Funzioni dedicate alla gestione dell'Object Snap Mode riconoscibili dal prefisso OSM_ nelle quali, una serie di suffissi specificano lo scopo della funzione: lettura G (get), scrittura S (set), salvataggio O (old), ripristino P (previous), disattivazione F (false), attivazione T (true). Ad esempio:

- *(OSM_OF) salva il valore corrente dell'Object Snap Mode e lo disattiva.*

[4] [5] [6] [7] Funzioni dedicate a varie letture riconoscibili dal prefisso GE_ seguito da suffissi che ne indicano la specificità: [4] di un punto PO e di una distanza dis forniti come dato di imput dall'operatore, [5] di un'entità grafica EN di tipo specificato già presente nel disegno corrente, [6] di una proprieta PR di tipo specificato da un entità fornita, [7] di una coppia di circonferenze 2CI o già presenti come entità grafiche nel disegno o tramite i loro centri e raggi. Ad esempio:

- *(GE_PR 5 (GE_EN 4)) chiede di selezionare una circonferenza e ne restituisce il raggio,*
- *(GE_PR 3 (GE_EN 2)) chiede di selezionare una linea e ne restituisce la lunghezza.*

[8] [9] Funzioni dedicate al disegno di entità grafiche riconoscibili dal prefisso DR_ seguito da suffissi che ne indicano la specificità: [8] di un'entità grafica EN di tipo specificato, [9] di un punto PO, di una linea LI, di una xlinea XL di una circonferenza CI. Ad esempio:

- *(DR_EN 1) e (DR_PO) disegnano un punto,*
- *(DR_EN 2) e (DR_LI) disegnano una linea.*

[10] [11] [12] [13] Funzioni dedicate a costruzioni geometriche che restituiscono un oggetto punto riconoscibili dal prefisso PO_ seguito da suffissi che ne indicano la specificità: [10] punto proiettato pro_ su una retta definita per due punti 2PO, [11] (in due versioni a b) punto simmetrico sim_ rispetto ad una retta definita per due punti 2PO, [12] punto medio med_ rispetto a due punti 2PO, [13] punto polare pol_ rispetto ad un punto un angolo e una distanza Pad.

Tribelon - Codici grafici 2 - Codice AutoLISP 3.lsp

```
;-------------------------------------------------------------------------------
;                                  Start Of File
;--[0]------------------------------------------------------------------------
(vl-load-com)
;--[1]------------------------------------------------------------------------
(defun MO_AcDo  ()          (vla-get-ActiveDocument (vlax-get-acad-object)))
(defun MO_Ut    ()          (vla-get-Utility        (MO_AcDo)))
(defun MO_MoSp  ()          (vla-get-ModelSpace     (MO_AcDo)))
;--[2]------------------------------------------------------------------------
(defun EN_U     (e)         (vla-Update e))
(defun EN_D     (e)         (vla-Delete e))
(defun EN_D3    (e1 e2 e3)  (mapcar 'EN_D (list e1 e2 e3)))
(defun EN_L     ()          (vla-Item   (MO_MoSp) (1- (vla-get-Count (MO_MoSp)))))
(defun EN_LU    ()                      (EN_U (EN_L)))
(defun EN_LD    ()                      (EN_D (EN_L)))
(defun EN_LD#   (num)       (repeat num (EN_D (EN_L))))
;--[3]------------------------------------------------------------------------
(defun OSM_G    ()          (vlax-variant-value (vla-GetVariable (MO_AcDo) "OSMODE")))
(defun OSM_S    (num)                   (vla-SetVariable (MO_AcDo) "OSMODE" num))
(defun OSM_O    ()          (setq *OSM-OLD* (OSM_G)))
(defun OSM_P    ()          (OSM_S *OSM-OLD*))
(defun OSM_F    ()          (if (<  (OSM_G) 16384) (OSM_S (+ (OSM_G) 16384))))
(defun OSM_T    ()          (if (>= (OSM_G) 16384) (OSM_S (- (OSM_G) 16384))))
(defun OSM_OF   ()          (OSM_O)  (OSM_F))
(defun OSM_OT   ()          (OSM_O)  (OSM_T))
;--[4]------------------------------------------------------------------------
(defun GE_PO    (PT §txt)   (vlax-variant-value (vla-GetPoint    (MO_Ut) PT §txt)))
(defun GE_Dis   (PT §txt)               (vla-GetDistance (MO_Ut) PT §txt))
;--[5]------------------------------------------------------------------------
(defun GE_EN    (# / §EN §txt EN_Sel PO_Sel)
 (cond ((= # 1) (setq §EN "AcDbPoint"    §txt "\nSelezionare un punto: "))
       ((= # 2) (setq §EN "AcDbLine"     §txt "\nSelezionare una linea: "))
       ((= # 3) (setq §EN "AcDbPolyline" §txt "\nSelezionare una polilinea: "))
       ((= # 4) (setq §EN "AcDbCircle"   §txt "\nSelezionare una circonferenza: ")))
 (while (/= (if EN_Sel (GE_PR 1 EN_Sel)) §EN)
        (vl-catch-all-apply 'vla-GetEntity (list (MO_Ut) 'EN_Sel 'PO_Sel §txt)))      EN_Sel)
;--[6]------------------------------------------------------------------------
(defun GE_PR    (# e)
  (cond ((= # 1)                    (vla-get-ObjectName  e))
        ((= # 2)                    (vla-get-Layer       e))
        ((= # 3)                    (vla-get-Length      e))
        ((= # 4)                    (vla-get-Angle       e))
        ((= # 5)                    (vla-get-Radius      e))
        ((= # 6) (vlax-variant-value (vla-get-Coordinates e)))
        ((= # 7) (vlax-variant-value (vla-get-Center      e)))
        ((= # 8) (vlax-variant-value (vla-get-StartPoint  e)))
        ((= # 9) (vlax-variant-value (vla-get-EndPoint    e)))))
;--[7]------------------------------------------------------------------------
(defun GE_2CI (fromMoSp / C1 POC1 C2 POC2 linC1C2 cir1 cir2)
  (if fromMoSp
      (progn (princ "\nSelezionare la prima circonferenza: ")    (setq cir1 (GE_EN 4))
             (princ "\nSelezionare la Seconda circonferenza: ")  (setq cir2 (GE_EN 4)))
      (progn (princ "\nSelezionare i dati per le due circonferenze: ")
             (setq C1       (GE_PO nil "\nSelezionare il primo centro: ")    POC1 (DR_PO C1)
                   C2       (GE_PO C1  "\nSelezionare il secondo centro: ")  POC2 (DR_PO C2)
                   linC1C2  (DR_LI C1 C2)
                   cir1     (DR_CI C1 (GE_Dis C1  "\nSelezionare il primo raggio: "))
                   cir2     (DR_CI C2 (GE_Dis C2  "\nSelezionare il secondo raggio: ")))
             (EN_D3 POC1 POC2 linC1C2)))                                    (list cir1 cir2))
;--[8]------------------------------------------------------------------------
(defun DR_EN  (# v1 v2 / en)
 (OSM_OF) (setq en (cond ((= # 1) (vla-AddPoint  (MO_MoSp) v1   ))
                         ((= # 2) (vla-AddLine   (MO_MoSp) v1 v2))
                         ((= # 3) (vla-AddXLine  (MO_MoSp) v1 v2))
                         ((= # 4) (vla-AddCircle (MO_MoSp) v1 v2))))       (OSM_P) (EN_LU) en)
;--[9]------------------------------------------------------------------------
(defun DR_PO  (PO      )   (DR_EN 1 PO   nil))
(defun DR_LI  (PO1  PO2)   (DR_EN 2 PO1  PO2))
(defun DR_XL  (PO1  PO2)   (DR_EN 3 PO1  PO2))
(defun DR_CI  (PO   Dis)   (DR_EN 4 PO   Dis))
;-[10]------------------------------------------------------------------------
(defun PO_pro_2PO (P1 P2 P3 / lin_a lin_b int_ab)
   (setq  lin_a  (DR_LI P2 P3)
          lin_b  (DR_LI P1 (PO_pol_Pad P1 (+ (GE_PR 4 lin_a) (/ pi 2)) 1))
          PO_int_ab (nth 0 (lisPO_int_2EN lin_a lin_b T)))              (EN_LD# 2) PO_int_ab)
;-[11a]-----------------------------------------------------------------------
(defun PO_sim_2PO (P1 P2 P3 / P1P P1S)
  (setq P1P (PO_pro_2PO  P1 P2 P3)
        P1S (PO_pol_Pad P1P (ang_2PO P1 P1P)(dis_2PO P1 P1P)))                          P1S)
;-[11b]-----------------------------------------------------------------------
(defun PO_sim_2PO (P1 P2 P3 / P1S)
  (setq P1S (vla-Mirror (DR_PO P1) P2 P3))                                  (EN_LD# 2) P1S)
;-[12]------------------------------------------------------------------------
(defun PO_med_2PO (P1 P2)      (PO_pol_Pad P1 (ang_2PO P1 P2) (/ (dis_2PO P1 P2) 2)))
;-[13]------------------------------------------------------------------------
(defun PO_pol_Pad (P a d)      (vla-PolarPoint (MO_Ut) P a d))
;-------------------------------------------------------------------------------
;                                  Midle Of File
;-[14]------------------------------------------------------------------------
```

3 | *Le routines proposte (seconda parte)*

[14] Funzioni dedicate ad effettuare misurazioni rispetto ad una coppia di punti 2PO: la prima dedicata a determinarne la distanza dis_ la seconda a determinarne la direzione come angolo ang_ rispetto all'asse x del sistema di coordinate di riferimento.
[15] [16] Funzioni dedicate all'estrazione di punti PO_ da una lista di lunghezza n contenente coordinate xyz ripetute in seguenza: [15] la prima (PO_nxyz lis pos) restituisce dalla lista lis uno specifico punto dalla posizione pos, [16] la seconda (lisPO_nxyz lis) restituisce dalla lista lis l'intera lista di punti.
[17] Funzione dedicata alla determinazione della lista dei punti lisPO_ d'intersezione int_ tra due entita grafiche 2EN.
[18] Funzione CIapo_2CI dedicata alla determinazione dei parametri della Circonferenza CI di Apollonio apo associata ed univoca rispetto ad una coppia di circonferenze 2CI opzionalmente disegnandola nel disegno corrente. Ad esempio:

- *(setq CAP (CIapo_2CI (GE_EN 4) (GE_EN 4) T) chiede due volte di selezionare una circonferenza (GE_EN 4) e disegnata la circonferenza di Apollonio associata T ne restituisce i parametri centro e punti diametrali nella variabile CAP.*

[19] Funzione RErad_2CI dedicata alla determinazione dei parametri della retta RE asse radicale rad associata ed univoca rispetto ad una coppia di circonferenze 2CI opzionalmente disegnandola come nel disegno corrente:

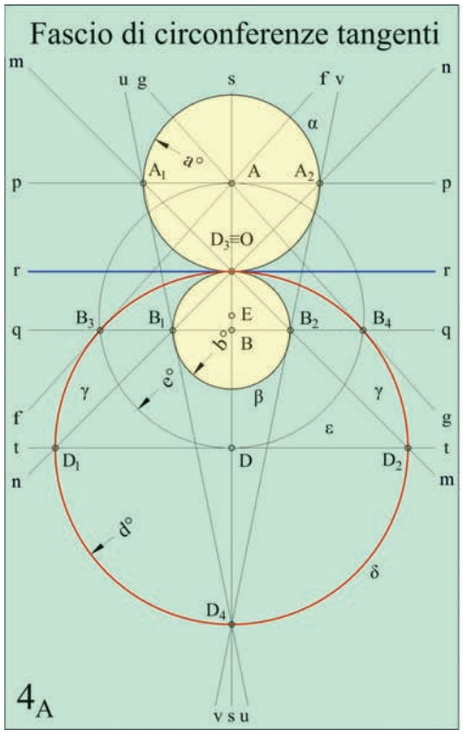
- *(setq RAD (RErad_2CI (GE_EN 4) (GE_EN 4) T) chiede due volte di selezionare una circonferenza (GE_EN 4) e disegnata la retta asse radicale associata T ne restituisce i parametri punto medio e punti estremi nella variabile RAD.*

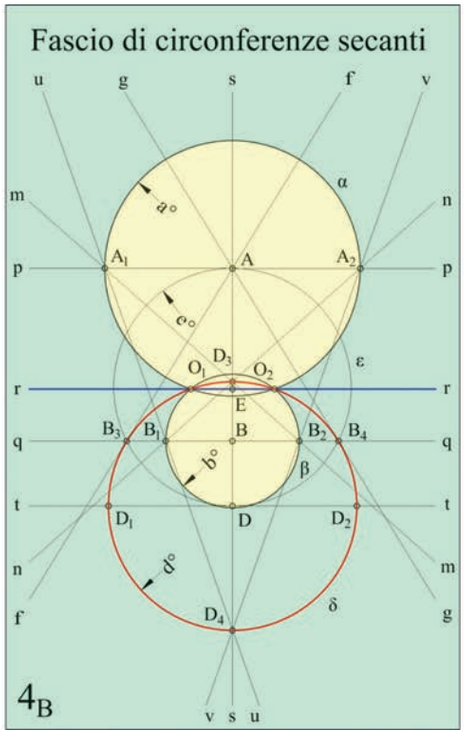
[20] Comandi per l'uso in AutoCAD delle funzioni in [18] e [19] con l'ausilio della funzione in [7]:

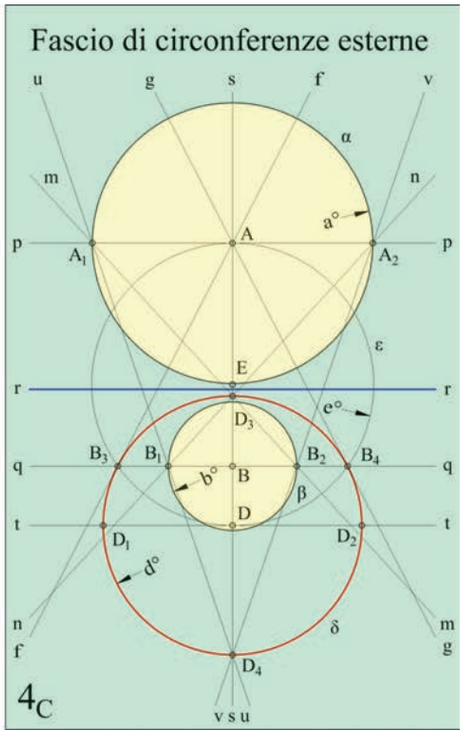
- *rad1 – comando per il disegno dell'asse radicale di due circonferenze selezionate come entità già presenti nel disegno,*
- *rad2 – comando per il disegno di due circonferenze e del loro asse radicale in base alla selezione di una coppia di punti quali centri e di due distanze quali raggi delle due circonferenze disegnate,*
- *apo1 – comando per il disegno della circonferenza di Apollonio associata a due circonferenze selezionate come entità già presenti nel disegno,*
- *apo2 – comando per il disegno di due circonferenze e della loro circonferenza di Apollonio in base alla selezione di una coppia di punti quali centri e di due distanze quali raggi delle due circonferenze disegnate.*

Tribelon - Codici grafici 2 - Codice AutoLISP 3.lsp

```
;-------------------------------------------------------------------------------
;                                  Midle Of File
;-[14]-----------------------------------------------------------------------
(defun dis_2PO (P1 P2 / dis)     (setq dis (GE_PR 3 (DR_LI P1 P2)))          (EN_LD) dis)
(defun ang_2PO (P1 P2 / ang)     (setq ang (GE_PR 4 (DR_LI P1 P2)))          (EN_LD) ang)
;-[15]-----------------------------------------------------------------------
(defun PO_nxyz (nxyz n)            (vlax-3d-point (nth (+ 0 (* 3 n)) nxyz)
                                                  (nth (+ 1 (* 3 n)) nxyz)
                                                  (nth (+ 2 (* 3 n)) nxyz)))
;-[16]-----------------------------------------------------------------------
(defun lisPO_nxyz (nxyz / n lPO)
  (setq n -1)
  (reverse (repeat (/ (length nxyz) 3)
                   (setq lPO (append (list (PO_nxyz nxyz (setq n (1+ n)))) lPO)))))
;-[17]-----------------------------------------------------------------------
(defun lisPO_int_2EN (EN1 EN2 ext / int var_int num_int)
   (setq int (vla-IntersectWith EN1 EN2 (if ext acExtendBoth acExtendNone)))
   (if (= (type int) vlax-vbEmpty)
       (setq num_int 0)
       (setq var_int (vlax-variant-value int)
             num_int (/ (1+ (vlax-safearray-get-u-bound var_int 1)) 3)))
   (if (>=  num_int 1)
       (lisPO_nxyz (vlax-safearray->list var_int))))
;-[18]-----------------------------------------------------------------------
(defun CIapo_2CI (CI1 CI2 draw / Cen1 Cen2 rag1 rag2 dirAR
                                 POA POB POC POD POapo1 POapo2 Cenapo)
   (setq Cen1    (GE_PR 7 CI1)
         Cen2    (GE_PR 7 CI2)
         rag1    (GE_PR 5 CI1)
         rag2    (GE_PR 5 CI2)
         dirAR   (+ (ang_2PO Cen1 Cen2) (/ pi 2)))
   (setq POA     (PO_pol_Pad Cen1 dirAR (+ rag1))
         POB     (PO_pol_Pad Cen1 dirAR (- rag1))
         POC     (PO_pol_Pad Cen2 dirAR (+ rag2))
         POD     (PO_pol_Pad Cen2 dirAR (- rag2)))
   (setq POapo1  (nth 0 (lisPO_int_2EN (DR_XL POA POD)
                                       (DR_XL POB POC) T))
         POapo2  (nth 0 (lisPO_int_2EN (DR_XL POA POC)
                                       (DR_XL POB POD) T))
         Cenapo  (PO_med_2PO  POapo1 POapo2))
   (EN_LD# 4)
   (list Cenapo POapo1 POapo2 (if draw (DR_CI Cenapo (dis_2PO Cenapo POapo1)))))
;-[19]-----------------------------------------------------------------------
(defun RErad_2CI (ci1 ci2 draw / int_ci12 int_num C1 C2 r1 r2 diC1C2 diC2C1 diAR
                                 PAR1 PAR2 PARC P1 P2 P3 P4 ci3 int_ci13 int_ci23 li12 li23)
   (setq int_ci12   (lisPO_int_2EN ci1 ci2 nil)
         int_num    (length int_ci12))
   (setq C1        (GE_PR 7 ci1)
         C2        (GE_PR 7 ci2)
         diC1C2    (ang_2PO C1 C2)
         diC2C1    (ang_2PO C2 C1)
         diAR      (+ diC1C2 (/ pi 2)))
   (cond ((= int_num 2) (setq  PAR1      (nth 0 int_ci12)
                               PAR2      (nth 1 int_ci12)
                               PARC      (PO_med_2PO PAR1 PAR2)))
         ((= int_num 1) (setq  PARC      (nth 0 int_ci12)
                               PAR1      (PO_pol_Pad  PARC diAR +1)
                               PAR2      (PO_pol_Pad  PARC diAR -1)))
         ((= int_num 0) (setq  P1        (PO_pol_Pad C1 diC1C2 (GE_PR 5 ci1))
                               P2        (PO_pol_Pad C2 diC2C1 (GE_PR 5 ci2))
                               P3        (PO_med_2PO  P1 P2)
                               P4        (PO_pol_Pad P3 diAR (dis_2PO P1 P3)))
                        (setq ci3        (DR_CI P4 (dis_2PO P1 P4)))
                        (setq int_ci13   (lisPO_int_2EN ci1 ci3 nil)
                              int_ci23   (lisPO_int_2EN ci2 ci3 nil))
                        (setq li13       (DR_XL (nth 0 int_ci13) (nth 1 int_ci13) )
                              li23       (DR_XL (nth 0 int_ci23) (nth 1 int_ci23) ))
                        (setq PAR1       (nth 0 (lisPO_int_2EN li13 li23 T))
                              PAR2       (PO_sim_2PO PAR1 C1 C2)
                              PARC       (PO_pro_2PO    PAR1 C1 C2))
                        (EN_LD# 3)))
   (list PARC PAR1 PAR2 (if draw (DR_XL PARC PAR1))))
;-[20]-----------------------------------------------------------------------
(defun c:rad1 ()  (setq lCI (GE_2CI  T ))  (RErad_2CI (nth 0 lCI) (nth 1 lCI) T))
(defun c:rad2 ()  (setq lCI (GE_2CI nil))  (RErad_2CI (nth 0 lCI) (nth 1 lCI) T))
(defun c:apo1 ()  (setq lCI (GE_2CI  T ))  (CIapo_2CI (nth 0 lCI) (nth 1 lCI) T))
(defun c:apo2 ()  (setq lCI (GE_2CI nil))  (CIapo_2CI (nth 0 lCI) (nth 1 lCI) T))
;-------------------------------------------------------------------------------
;                                  End Of File
;-------------------------------------------------------------------------------
```







4 | *Una coppia di circonferenze α β di centro A B e raggi a° b° sono circonferenze generatrici di un fascio di circonferenze, nel quale è possibile determinare un asse radicale r ed una circonferenza di Apollonio δ di centro D e raggio d°, associata in maniera univoca a ciascuna coppia di circonferenze del fascio, ivi comperse le due generatrici. Per approfondire l'aspetto informatico relativo alla programmazione del codice delle due procedure, i loro algoritmi sono rispettivamente descritti ai punti [19] (RErad_2CI) e [18] (CIapo_2CI) della didascalia di fig. 3 e nel testo a lato del paragrafo Un esempio di codice; si rimanda ad essi per approfondimenti.*
Nelle tre immagini della figura i tre casi di possibile configurazione per le due circonferenze generatrici e per i fasci ad esse associati:
- *fascio di circonferenze tangenti (4A)*
- *fascio di circonferenze secanti (4B)*
- *fascio di circonferenze esrterne (4C)*

Dal punto di vista grafico, per la descrizione della costruzione dell'asse radicale e della circonferenza di Apollonio, si rimanda per approfondimenti ai testi (1) (2) in bibliografia. Una possibile costruzione della circonferenza di Apollonio, identica nei tre casi, è la seguente: si determinano per entrambe le circonferenze generatrici α β i punti diametrali A_1 A_2 sulla retta p e B_1 B_2 sulla retta t rette entrambe ortogonali all'asse centrale s per i due centri A B; il punto D_3 d'intersezione delle rette m (A_1 B_2) n (A_2 B_1) ed il punto D_4 d'intersezione delle rette u (A_1 B_1) v (A_2 B_2) saranno punti diametrali della circonferenza di Apollonio δ di centro D punto medio (D3 D4) .

Funzione CIapo_2CI

Sintassi: (CIapo_2CI cir1 cir2 draw)
Variabili in entrata: **cir1** e **cir2** valori richiesti VLA-OBJECT di tipo IAcadCircle; **draw** valori richiesti T/nil (vero/falso).
Variabili private: Cen1, Cen2, rag1, rag2, dirAR, POA, POB, POC, POD, POapo1, POapo2, Cenapo.
Vengono estratti (GE_PR) dalle due circonferenze fornite centri **Cen1 Cen2** e raggi **rag1 rag2** e si determina la direzione dell'asse radicale **dirAR** come ortogonale alla direzione in radianti dei due centri (ang_2PO) aggiungendovi il valore (/ pi 2) = 90°.
Si determinano (PO_pol_Pad) dai due centri due coppie di punti **POA POB** e **POC POD** in direzione dirAR a distanza ± rag1 e ± rag2.
Si determinano i punti: **POapo1** di intersezione (lisPO_int_2EN) tra le due rette (DR_XL) per le coppie di punti POA POD e POB POC; **POapo2** analogamente per le coppie di punti POA POC e POB POD; **Cenapo** loro punto medio.
Si predispone una lista (list) di dati in uscita: Cenapo, la coppia POapo1 POapo2 quali centro e punti diametrali della Circonferenza di Apollonio cir1 cir2 e, se (if) draw=T, l'entità eventualmente disegnata (DR_CI).

Bibliografia

G. Anzani, *Fasci di circonferenze*, Lulu 2023.

G. Anzani, *Algoritmi di geometria descrittiva in AutoLISP su punti rette e piani*, Lulu 2018.

T. Bousfield, *A practical guide to Autocad AutoLISP*, Longman 1998.

E. C. Jeffrey, *Programmare AutoCAD con VBA*, Mondadori Informatica 2002.

R. J. Krawczyk, *The codewriting, New Yorkworkbook – Creating computational architecture in AutoLISP*, Princeton Architecture Press 2009.

K. Standiford, *AutoLISP to Visual LISP: design solution for AutoCAD*, Thomson Learning (Autodesk Press), Canada 2001.

R. Togores Fernández, C. Otero González, *Programación en AutoCAD con Visual LISP*, Mc Graw Hill, Madrid 2003.

R. N. Togores, *AutoCAD expert's Visual LISP*, Createspace Independent Pub, 2012.