Developing Associative Functionality in AutoCAD

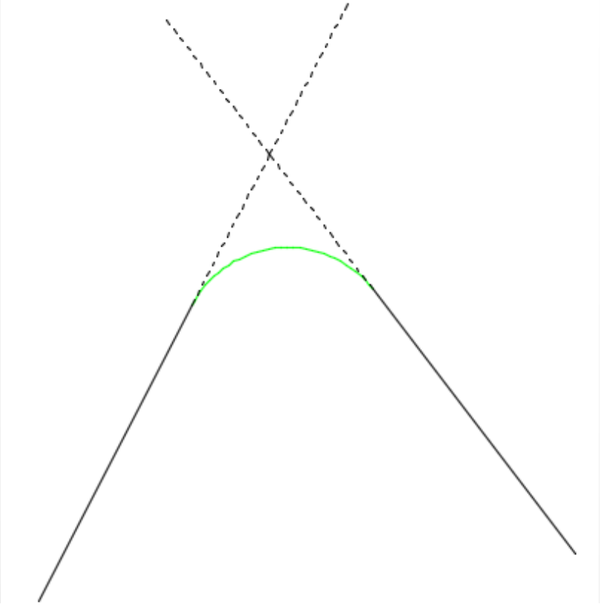
Jiri Kripac

Senior Software Architect

jiri.kripac@autodesk.com

Class Summary

- Associative applications represent relations between objects and maintain Design Intent in AutoCAD drawings/models
- Drawings/models are “intelligent”, not just collection of static “dumb” geometry
- Associative Framework is the foundation to build such applications
- Demo and step-by-step code example: Associative Fillet



Key Learning Objectives

At the end of this class, you will:

- Understand concepts and building blocks of the AutoCAD Associative Framework
- Understand source code of a complete and realistic sample application
- Be comfortable writing your own applications using the Associative Framework
- Have the sample application source code available to you so that you can use it as a template

Agenda

Associative Framework:

- Core building blocks: Action, Dependency, Action Parameter, Network
- Variables and expressions
- Network evaluation, evaluation order
- Change notification, transitive change propagation
- Deep cloning, dragging

Associative Fillet sample application:

- Demo of the functionality
- Structure of the application
- Complete source code walk-through
- Review of related ObjectARX header files

Building Applications in AutoCAD

- AutoCAD is a general, flexible, domain-independent, 2D/3D platform
- Custom objects derived from AcDbObject and AcDbEntity base classes
- General reactor notification mechanism

However:

- Higher-level functionality not directly provided
  (verticals built their own higher-level layers)
- In particular, no consistent and unified mechanism was available to represent relations between objects and to perform automatic updates after changes

  (“intelligent” models/drawings preserving Design Intent)

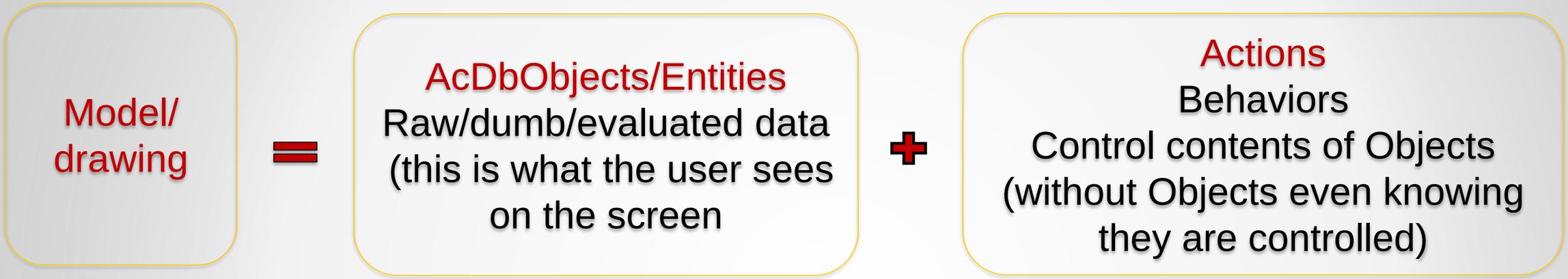
AutoCAD Associative Framework

Associative Framework

- Uniformly represents and maintains relations between Objects
- Transitively updates everything that needs to be updated when some parameters in the drawing/model change
- Relations are represented as hierarchical Networks of Actions, Objects, and Dependencies between them
- General, uniform, domain independent solution
- Part of AcDb

Used to implement all new AutoCAD associative functionality such as 2D Constraints, Associative Dimensions, Associative Array, Model Documentation, Associative Surface Modeling, etc. Used by verticals

Separation of Data and Behaviors



- Enables to *mix-and-match* Actions and Objects
- New Actions can extend behaviors of existing Objects
- For instance: We do not have a new constrained Line or Circle in AutoCAD, they are still the good old Lines and Circles

Separation from Objects

- Always interacting with (AcDb)Objects via protocol extensions, never directly
- New Objects plug themselves in by revealing properties that the framework expects
- Done from the outside by the Object exposing the required protocol extensions. The Object's code does *not* need to be modified
- It then all starts working with the new Objects:
  - Lines, Arcs or Walls become constrainable,
  - Solids become associatively dimension-able,
  - etc.

Actions Follow Objects

When a set of Objects with constraints or other Actions is cloned, the appropriate constraints, Variables, expressions and other Actions are cloned along with the Objects, and updated to reflect the new context

Core Building Blocks

Action	Behavior/”intelligence”
Action Parameter	Data of the Action
Dependency	Associativity/relations
Network	Associative model
Object	Regular AutoCAD AcDbObjects/AcDbEntities keep the evaluated state of the model

Action - Representing Behavior

- AcDbAssocAction class
- The evaluate() method exercises a custom behavior that takes input, performs a custom operation and produces output (mainly by changing properties of AcDbObjects in the drawing)
- Owns Dependencies that represent the dependency of the Action on arbitrary Objects
- May use multiple Objects and modify multiple Objects
- One Object may be modified by multiple Actions
- Often defines a “parent-child” relationship, e.g. a blend surface depends on edges of the input surfaces, but not vice versa

Action Body – Implementing Custom Behavior

- AcDbAssocActionBody base class to derive custom Action behaviors
- Owned by the Action
- The primary method to override and implement is AcDbAssocActionBody::evaluateOverride()
- Custom Actions not directly derived from AcDbAssocAction to better handle “zombies” (i.e. the situation when the application implementing the class is not present)
- AutoCAD supports SaveAs up to R14. Handling “zombies” is critical (and *very painful and time consuming*!)

Action Parameter - Keeping Custom Data

- Owned by the Action
- May keep the data in a variety of forms and provides the resulting value to the Action
- Examples:
  - AcDbAssocFace/Edge/VertexActionParam
  - The edge (curve) may be obtained from an edge of a referenced solid or surface, be a referenced line or circle entity, segment of a polyline, constant AcGeCurve3d, etc.
  - Value Action Parameter – numerical value, possibly defined by an expression referencing other Objects

Dependency - Representing Associativity/Relations

- AcDbAssocDependency class
- Keeps information about an Action depending on a property of an Object
- Owned by an Action. Attached to the Object as a Persistent Reactor
- Filters-out irrelevant Object change notifications
- Read and write Dependencies:

  Read-only – Uses the value of Object's property

  Write-only – Modifies the value of Object's property

  Read-write – Both uses and modifies Object's property

- The list of Dependencies on the Object is ordered. Dependencies explicitly specify their order

Dependency on Named Property

- AcDbAssocValueDependency depends on a simple named property (double, int, point, string, etc.) of an Object
- Objects reveal their named properties by exposing AcDbAssocValueProviderPE protocol extension

Dependency on Topological Subentity

- AcDbAssocGeomDependency depends on a topological subentity (face, edge, vertex) of an Object
- Objects reveal their topological subentities by exposing AcDbAssocPersSubentIdPE protocol extension
- Simple Objects like Lines and Arcs have a fixed set of subentities. Objects like Solids/Surfaces have a variable and changing set of subentities
- AcDbAssocPersSubentId – base class for persistent subentity identifiers

Mapping:

- Current (transient) AcDbSubentId → AcDbAssocPersSubentId
- AcDbAssocPersSubentId → Current (transient) AcDbSubentId(s)

Dependency Scenarios Examples

Associative Dimension Action (also used in Model Documentation):

- Read-only Geom Dependencies on subentities (e.g. vertices, edges) of the dimensioned entities
- Write-only Dependency on the controlled AcDbDimension object

Constraint Group Action:

- Read-write Geom Dependencies on constrained subentities of entities
- Read-only Value Dependencies on Variables defining dimension values
- Read-write Dependencies on dimension entities

Variables and Expressions

- AcDbAssocVariable keeps name, expression and value
- May depend on other Objects providing a value (expose AcDbAssocValueProviderPE protocol extension), such as on other Variables or properties of other Objects
- Is an Action, has Dependencies on other Objects, evaluates in the correct order, is part of a Network as any other Action
- PARAMETERS and -PARAMETERS commands to access and edit Variables and expressions
- Using Object properties in expressions (undocumented):

  2*ObjectHandle_HANDLE.Radius + 0.1

Network - Representing Associative Model

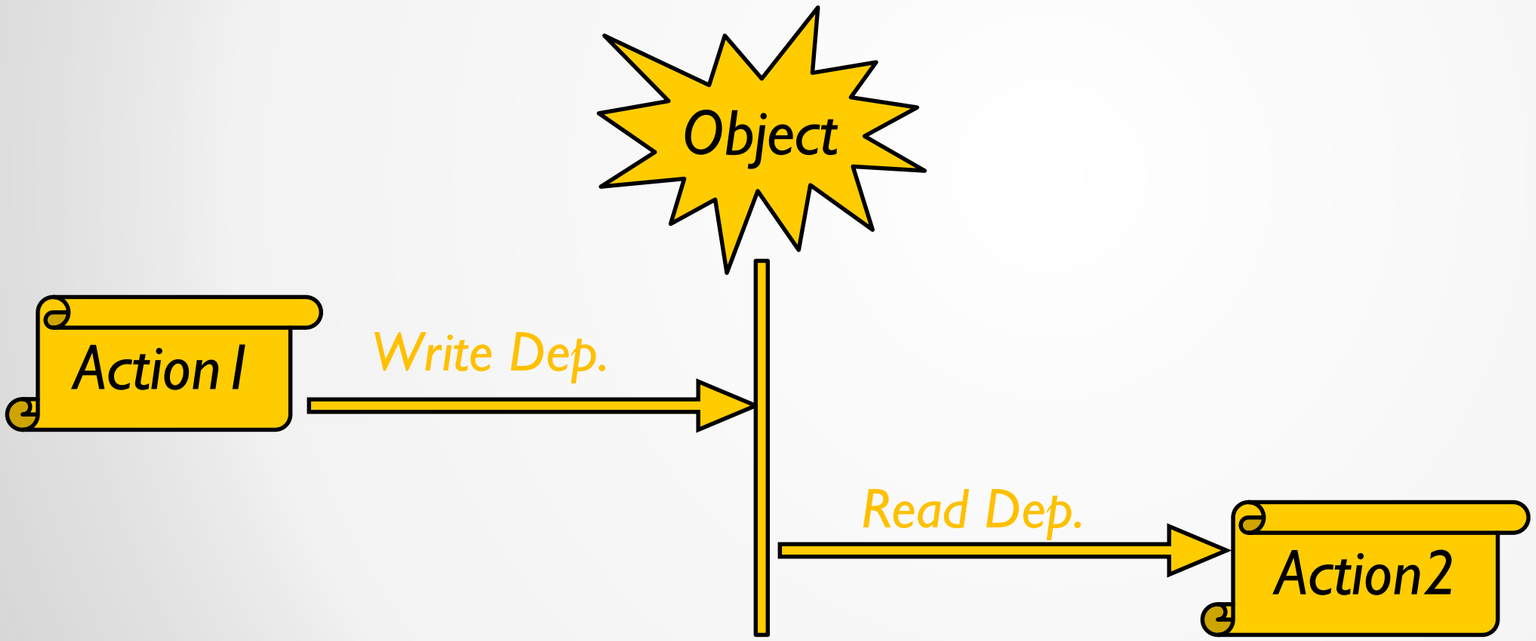
- AcDbAssocNetwork class owns a set of Actions (but does not own the Objects that the Actions control)
- Network itself is an Action, thus allowing to create hierarchical Networks
- A single top-level Network for the whole AcDbDatabase. Sub-Networks for AcDbBlockTableRecords. Other arrangements permitted

Network Evaluation

- The evaluate() method of a Network collects all Actions that need to be evaluated and evaluates them in the correct order
- The evaluation synchronizes the model/drawing, mainly by updating the Objects, so that the relations between the Objects are preserved and the design intent is maintained
- The evaluation is always on explicit request (either by AutoCAD or by the client code), never automatic
- AutoCAD evaluates the top-level Network only on document lock change (command begin/end) and after every drag sample

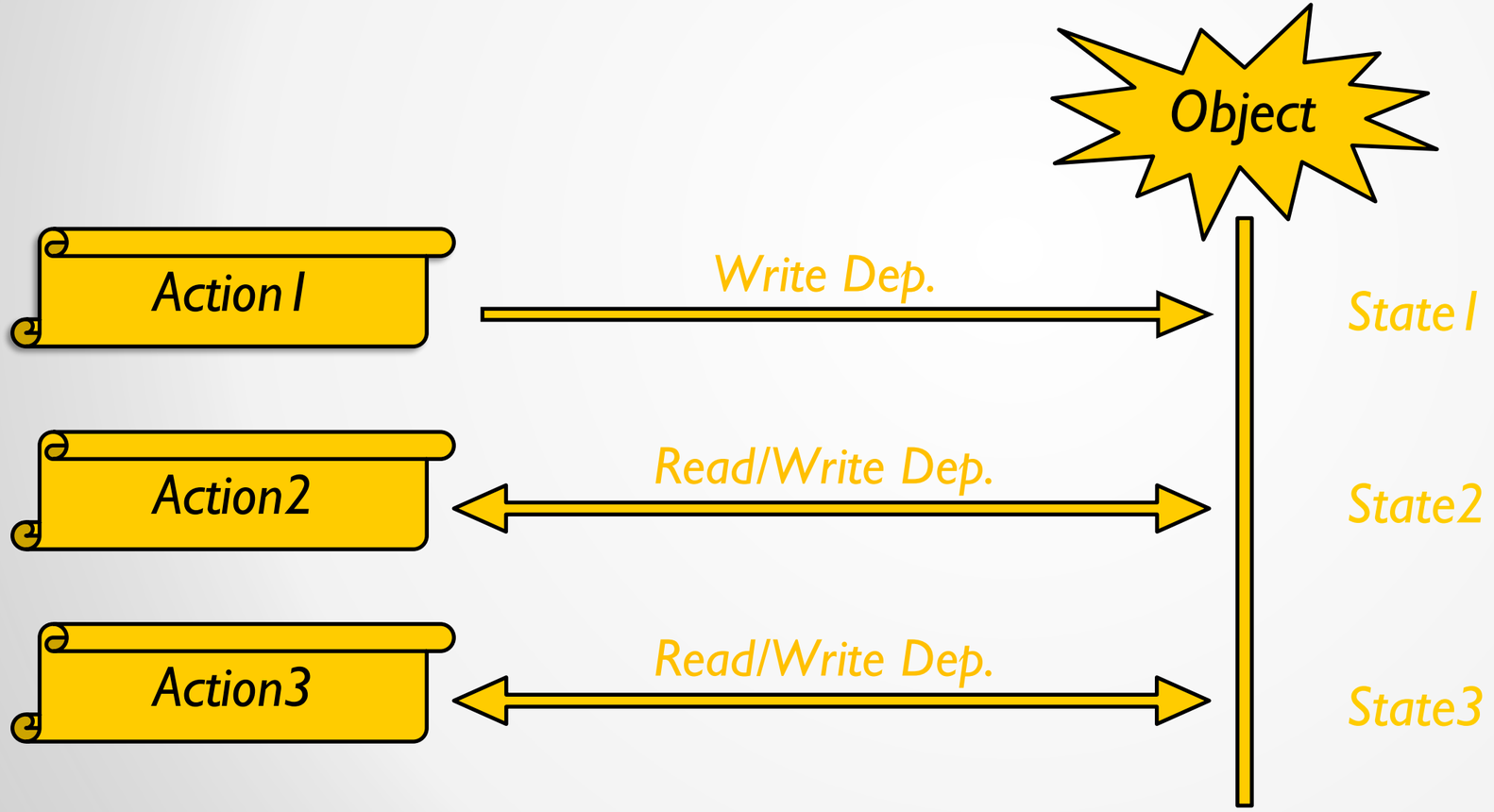
Evaluation Order

Action1 modifies Object and Action2 uses it. It establishes ordering that Action1 needs to be evaluated before Action2



Evaluation Order

An Object has multiple Object States. To evaluate an Action, the Object must be in the correct State



Change Notification

Actions, Dependencies and Action Parameters have an evaluation status data member:

```
enum AcDbAssocStatus
{
    kIsUpToDateAssocStatus,
    kChangedDirectlyAssocStatus,
    kChangedTransitivelyAssocStatus,
    kChangedNoDifferenceAssocStatus,
    kFailedToEvaluateAssocStatus,
    kErasedAssocStatus,
    kSuppressedAssocStatus,
};
```

Change Notification

When an Object is modified:

- Each Dependency on the Object checks if the change is relevant to that Dependency. If not, nothing happens. If yes, the Dependency changes its Status to kChangedDirectlyAssocStatus
- The Dependency notifies the Action owning the Dependency. The Action changes its own Status to kChangedDirectlyAssocStatus
- The Action notifies the owning Network. The Network changes its own Status to kChangedDirectlyAssocStatus. This continues up to the top-level Network
- *No immediate evaluation happens*

Transitive Change Propagation

- Happens just before the top-level Network is requested to be evaluated
- Transitively changes Status of other Dependencies and Actions, obtaining transitive closure of all Actions that need to be evaluated
- The default algorithm is based on the evaluation order defined by the read and write Dependencies of the Actions and by the order of the dependencies on Objects
- Actions may control the change propagation and override the default algorithm

Ordered Action Evaluation

- Finds all Actions that need to be evaluated and can be evaluated, and evaluates them
- Actions can control the evaluation order using their evaluation priority
- The Status of evaluated Actions changes to kIsUpToDateAssocStatus or kFailedToEvaluateAssocStatus
- Evaluating an Action may unblock evaluation of other Actions
- This process continues until all Actions are evaluated

Deep Cloning and the Associative Framework

- Numerous cloning scenarios in AutoCAD:

  Copy, array, explode, block, wblock, insert, bedit, refedit, <Ctrl>-C/X/V, xref-bind, etc.

- When an Action has Dependencies on cloned Objects, it is notified and can request more Objects to deep clone (including itself)

- Actions of cloned Objects are notified to do post-processing after the deep clone

- The framework tries to automatically handle creating Networks, adding cloned Actions to correct Networks, multiple Dependencies on Objects, cloning in the same database or cross-database, automatic Variable renaming, cloning/not cloning referenced Variables, etc.

Dragging and the Associative Framework

- When entities that have Dependencies on them are dragged, the top-level Network is evaluated on every drag sample
- For performance reasons, each Action may decide whether it does or does not want to evaluate (i.e. its evaluation is a no-op)
- AutoCAD dragging loop: Clone, modify, display, delete
- Action evaluation is transparently redirected to non-database resident clones using AcDbAssocObjectPointer
- It takes AcDbObjectId of a database-resident entity and returns a pointer either to that database-resident entity or to its non-database resident clone
- *Application code that uses the Associative Framework does not need to do anything; it all happens automatically*

Sample Application: Associative Fillet

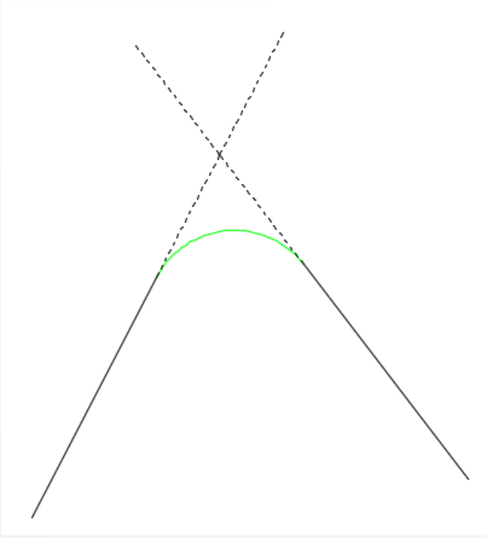
Associative Fillet

A new AssocFilletActionBody

Depends on two input curve entities (done generally, as dependencies on two edge subentities of two entities)

Controls an AcDbArc entity that is the fillet between the two edge subentities

Optionally trims/extends the input entities to the fillet arc



Associative Fillet

The fillet radius can be controlled by an expression referencing parameters

The options whether to trim/not-trim the input entities can also be controlled by expressions

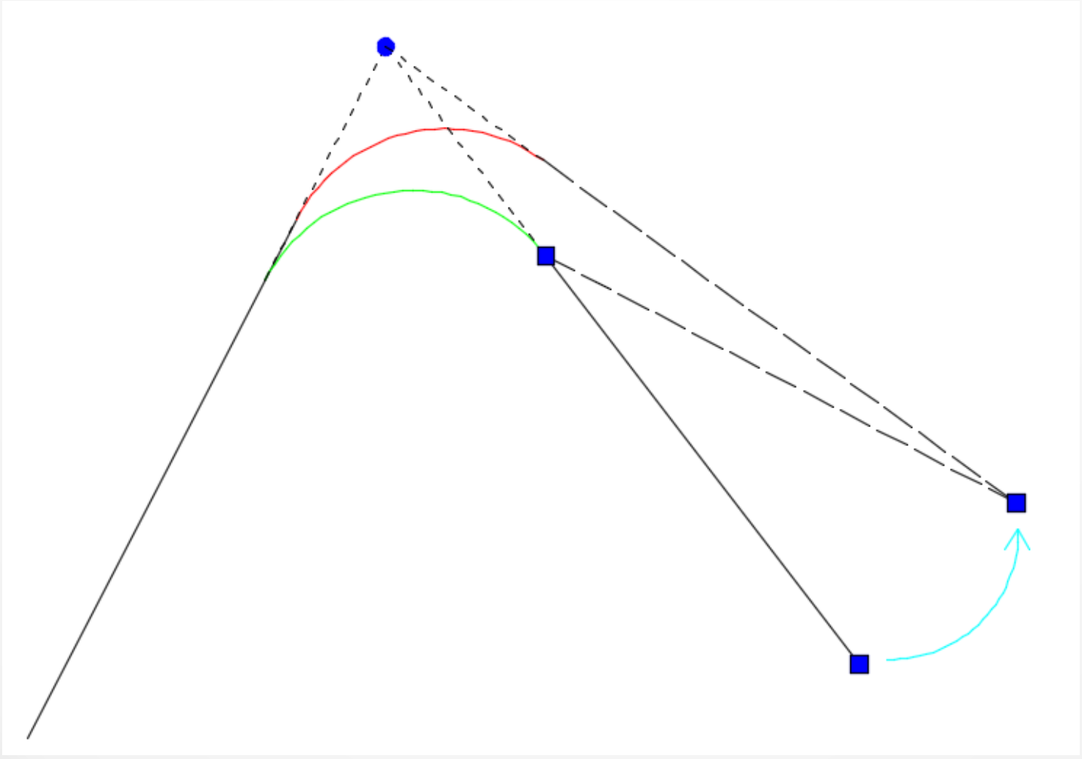
When the input curves intersect at multiple intersections, it tries to keep the fillet at the same intersection

When the input entities are cloned, the Associative Fillet Action and the AcDbArc are also cloned

When at least one of the input entities is erased, the Associative Fillet Action and the AcDbArc are erased

Modifying the Input Curve Dragging Behavior

When the input curve is linear and its end grip point is being dragged, the line is dragged as if the other endpoint of the line was at the apparent intersection of the two input curves, not at ending at the fillet arc



Live Demo

- Command: APPLOAD, load the AssocFillet.arx application
- Create two intersecting lines in AutoCAD
- Command: PARAMETERS, create new parameters Trim1 and Trim2 with value 0
- Command: ASSOCFILLET, follow the prompts, enter non-zero radius and use Trim1 and Trim2 as the expressions to control whether to trim/extend the input lines
- Change the color of the fillet arc to Green
- Command: PARAMETERS, change value of Trim1 to 1, and then Trim2 to 1, see what happens
- Select end grip point on one of the lines, drag the grip point. See how the dragged line behavior changes due to the fillet, the fillet keeps updating, and the lines keep re-trimming
- Command: PARAMETERS, create a new parameter named A, assign it a value, such as 3.0, create a new parameter named Radius, assign it an expression, such as “A/2"
- Command: ASSOCFILLET, select the associative fillet arc, change radius to be an expression, such as “FilletRadius=Radius+0.1“
- Command: COPY, select the two input lines, but not the associative fillet arc, make two copies. See that the associative fillet arc has been copied and stays associative
- Command: PARAMETERS, change value of parameter A, see that all associative fillets update

Live Demo

- Command: ELLIPSE, create two intersecting ellipses, one horizontal, another vertical
- Command: ASSOCFILLET, create a fillet at one of the 16 possible positions
- Grip-edit one of the ellipses, see that the associative fillet stays
- Move one ellipse, see that that associative fillet stays
- Draw two polylines with several segments
- Command: ASSOCFILLET, select two segments of the two polylines using <Ctrl> selection
- Drag one polyline, see that the associative fillet updates
- Delete and insert some polyline segments. See that the associative fillet stays attached to the same segments even if the number of the segments changed – the fillet stores AcDbAssocPersSubentIds of the polyline segments
- Command: EXTRUDE, select the input lines and the fillet arc, specify extrusion height by expression, such as “3*A”
- Command: PARAMETERS, change value of parameter A. See that the associative fillet updates, followed by the update of the extrusion surface that depends on the three entities (the two lines and the fillet arc). The input lines and the fillet arc have two actions attached to them: The associative fillet action as well as the extrusion action
- Drag one of the lines. See that the fillet as well as the extrusion surface update

Live Demo

- Command: ASSOCFILLET, select a fillet, change radius to 0.0 – becomes associative trim
- Command: ERASE, select one of the lines. See that the associative fillet arc has also been erased
- Command: UNDO to bring the associative fillet back
- Command: PARAMETERCOPYMODE, enter 4 to copy all referenced parameters
- Command: WBLOCK, select one pair of input lines
- Open a new drawing
- Command: INSERT, choose the drawing created by the WBLOCK command, make several inserts
- Command: EXPLODE, select a block reference
- Grip-edit one line of the exploded block reference. See that the associative fillet has been preserved
- Command: PARAMETERS: See the parameters the associative fillet depends on are there

Source Code Walk Through

AssocFilletActionBody.h

Derived from AcDbAssocActionBody

Methods:

- Overridden AcDbAssocActionBody protocol
- Overridden AcDbObject protocol
- Application-specific protocol

Data:

- Mostly kept in Action Parameters (serialized automatically by the AcDbAssocAction class)
- Only one explicit data member (mFilletConfig)

AssocFilletActionBody.cpp

get/setRadius():

- Data stored in a value action parameter (that may keep an expression)

get/setInputEdge():

- Data stored in AcDbAssocEdgeActionParam. May reference a whole entity (such as an AcDbLine) or an edge subentity (e.g. a segment of an AcDbPolyline)
- AcDbEdgeRef used to pass around the edge subentity data
- Using AcDbAssocObjectPointer so that this method can be called during dragging

is/setTrimInputEdge()

- Data stored in a value action parameter
- Also needs to synchronize the Dependency to be read-only or read-write

getFilletArcId():

- The fillet AcDbArc referenced by an AcDbAssocDependency

AssocFilletActionBody.cpp

evaluateOverride():

- Regular or relaxed evaluation
- Erasing the Action if some of its Dependencies broken
- Using AcDbAssocObjectPointer so that this method can be executed during dragging
- If successful, modifies the AcDbArc entity and (optionally) trims/extends curves of the input edge subentities
- If the evaluation cannot be completed, reports an error via AcDbAssocEvaluationCallback::setActionEvaluationErrorStatus() and sets the status of the Action as kFailedToEvaluateAssocStatus. Otherwise sets the Action status as kIsUpToDateAssocStatus

addMoreObjectsToDeepCloneOverride/postProcessAfterDeepCloneOverride():

- Adds the Action and fillet AcDbArc to be cloned if input geometries are cloned
- Moves the cloned fillet AcDbArc to the same BTR as the first cloned input edge

AssocFilletActionBody.cpp

createAndPostToDatabase():

- Pseudo-constructor
- Creates the Action, Action Body, sets them up, adds them to database and to a Network

dwg/dxfIn/OutFields():

- Only mFilletConfig needs to be explicitly serialized
- Version handling

Associative Fillet Dragging – Flow of Execution

- The user selects one input entity and drags its grip point
- On every drag sample the dragger makes a non-database-resident clone of the dragged entity and modifies it, instead of modifying the original database-resident entity
- The Associative Fillet Action is notified because it has a Dependency on the dragged entity and its evaluateOverride() method is called (notice that even if the non-database-resident clone is modified but the Dependency is on the non-changed original database-resident entity, the Dependency is still notified)
- The Action opens the input entities for read. In case of the dragged entity, the Action receives a pointer to a non-database-resident clone that the dragger created
- The Action calculates new geometry of the fillet arc and of both input entities

Associative Fillet Dragging – Flow of Execution

- The Action opens all entities (the two input entities and the fillet arc) for write in order to modify them. *AcDbAssocObjectPointer* is used
- The dragger makes non-database-resident clones of the *second (non-dragged) entity* and of the *fillet arc* and returns them to the Action
- The Action modifies geometries of *all three entities* (it modifies non-database-resident clones, not the original entities)
- After the evaluation of all Actions is finished, the dragger draws all modified entities (which are non-database-resident clones), deletes the clones, and performs undo (on the non-graphical data)
- On the last drag sample, the operation is performed with the original database-resident entities
- The implementer of the Associative Fillet does not need to do anything special to support dragging; it all happens automatically

AssocFilletConfig Class

Keeps information about in which of the 4 quadrants around the intersection of the two curves to place the arc

Does the adjustment of the input curves when they are linear and being dragged

Keeps information about which of the possible multiple intersections between the two input curves to use to place the fillet. Uses a configuration flag and parametric positions of the intersection on both curves

When the input curves change and the fillet arc is re-evaluated, it finds the intersection with the same configuration and closest (in parametric space) to the original one. Notice that the curves may be periodic or closed, so the closest distance calculation needs to take this into account

Command-Line UI

New ASSOCFILLET command

Prompts to select the first input entity or edge subentity, or an existing Associative Fillet arc

Then it goes to either create a new Associative Fillet or to edit an existing one

Command-Line UI: Creating a New Associative Fillet

If the first selected entity or edge subentity is a general entity (not an Associative Fillet arc), prompts for the second input entity or edge subentity, fillet radius, and whether to trim/extend the first and the second input

The radius and whether to trim/extend the input edges can be specified by expressions

Creates a new Associative Fillet Action by calling AssocFilletActionBody::createAndPostToDatabase() pseudo-constructor

Evaluates the Network by calling AcDbAssocManager::evaluateTopLevelNetwork()

Checks if the newly created Associative Fillet Action evaluated successfully. If not, erases it

Command-Line UI: Editing an Existing Associative Fillet

If the first selected entity is an existing Associative Fillet AcDbArc entity, it indicates that an existing Associative Fillet is to be edited

Obtains the AssocFilletActionBody from the AcDbArc entity

Prompts for the fillet radius, and whether to trim/extend the first and the second input edge

As defaults uses the values obtained from the selected Associative Fillet

Sets new values in the existing AssocFilletActionBody

The changed Action is then automatically evaluated at the end of the command

Main Associative Framework Header Files

Base classes to derive from:

- AcDbAssocActionBody.h
- AcDbAssocPersSubentIdPE.h

Classes to use:

- AcDbAssocFace/Edge/VertexActionParam.h
- AcDbAssocDependency.h
- AcDbAssocGeomDependency.h
- AcDbAssocManager.h
- AcDbAssocAction.h
- AcDbAssocGlobal.h

More Information

Feel free to contact me at jiri.kripac@autodesk.com

I work in San Rafael, California, Autodesk office

Associative Fillet source code posted on public ADN GitHub:

https://github.com/ADN-DevTech/Associative-Fillet-sample-application

Documentation:

- Associative Framework section in the AutoCAD ObjectARX Developer Guide
- ObjectARX Reference Guide (the same documentation as in .h files)
- All class names start with “AcDbAssoc” prefix