

[Student Notes:](#)



CATIA Training

Foils

CAA V5 for CATIA

Getting Started

Version 5 Release 16
November 2005

EDU-CAA-EN-CAG-AF-V5R16

[Student Notes:](#)

Table of Contents (1/2)

Native CATIA V5 Openness Positioning	p.4
CAA V5 Development Environment Basics	p.33
CAA V5 Object Modeler: Interface / Implementation Design Pattern	p.67
CAA V5 Object Modeler: Extension Mechanism	p.96
CAA V5 Object Modeler: Link between Interface and implementation	p.104
CAA V5 Object Modeler: Component	p.110
CAA V5 Object Modeler: Object Life Cycle Management	p.116
CAA V5 Specification Modeler	p.131
CAA V5 Mechanical Modeler: Introduction	p.139
CAA V5 Mechanical Modeler: The “Root” Features	p.150
CAA V5 Mechanical Modeler: The “GeometricalElement3D” Features	p.162

[Student Notes:](#)

Table of Contents (2/2)

■ CAA V5 Mechanical Modeler: The “Constraint” Features	p.172
■ CAA V5 Mechanical Modeler: The “Axis System” Features	p.175
■ CAA V5 Drafting and Tolerancing	p.179
■ CAA V5 ApplicationFrame	p.190
■ CAA V5 Dialog	p.206
■ CAA V5 DialogEngine	p.226
■ CAA V5 Resources	p.253
■ CAA V5 Visualization	p.261
■ CAA V5 Geometric Modeler (CGM)	p.279
■ CAA V5 Mechanical Modeler: Selection Object	p.299
■ CAA V5 Knowledgeware: Parameters and Relations	p.317
■ CAA V5 Product Structure & Assembly	p.338
■ CAA V5 Development Environment: Quality Control	p.353
■ CAA V5 Administration	p.377
■ Annexes	p.396

Student Notes:

Native CATIA V5 Openness Positioning

You will learn what are the different techniques and tools available to customize CATIA V5 and when to use which one.

- Application typology
- Some Case Studies
- Summary

Student Notes:

Application typology

- Create application
 - ◆ Automate Design Best Practices
 - ◆ Define some new user interface
- Extend the V5 PPR
 - ◆ Enrich the V5 data model
- Connect an external application to V5



Technological Choices

[Student Notes:](#)



Create Application



Using interactive techniques:

- BKT – Business Process Knowledge Template
 - ◆ Automate Design Best Practices
- KWE – Knowledge Expert:
 - ◆ Check Corporate Rules



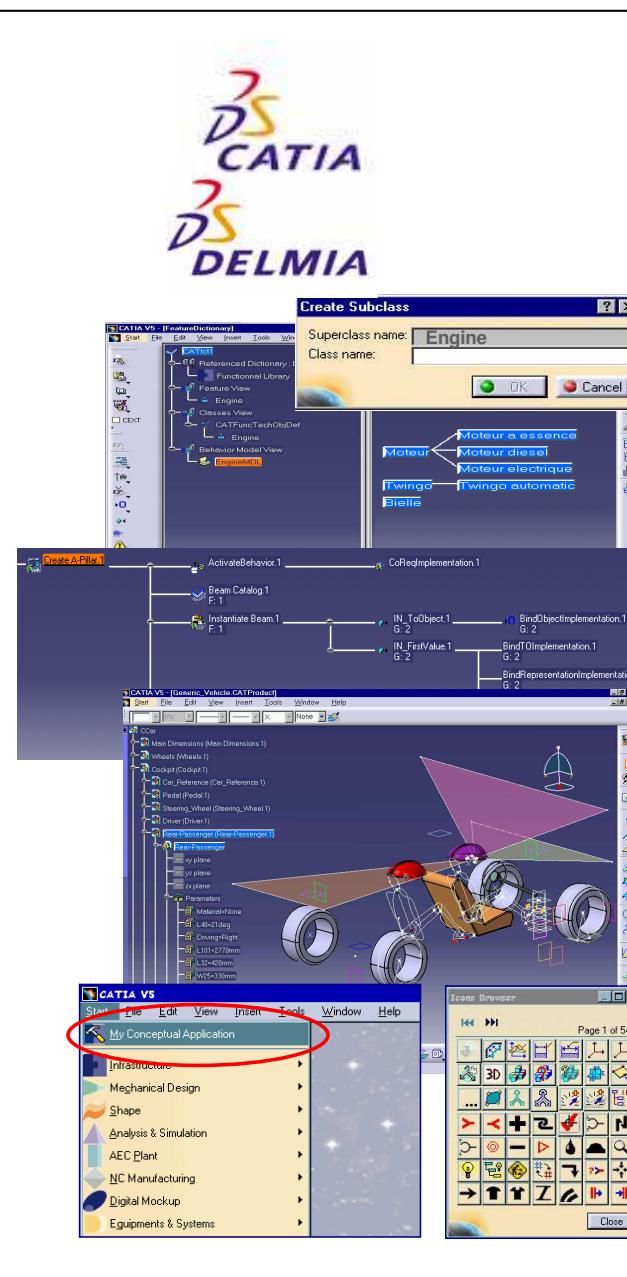
Using programming techniques:

- VB macros using V5 Automation API
 - ◆ Define simple user interface to manage V5 objects
- CAA
 - ◆ Define advanced user interface to manage V5 objects



Business Process Knowledge Template

- ❖ Capture and reuse corporate engineering and design processes
- ❖ Generate interactively customized vertical applications
 - ◆ Application technological objects
 - Attributes
 - Types
 - Rules
 - ◆ Application logic: behavioral model
 - Design, Engineering, Manufacturing,
 - Decision Support Tasks, etc...
 - Reuse of all V5 Tools
 - ◆ User Interface tools and technological object environment
 - Workbench Generator
 - Icon Editor
- ❖ The application is described in a directory including:
 - ◆ A feature catalog
 - ◆ Resource files



[Student Notes:](#)

Student Notes:



BKT Characteristics

Scope of Work

All V5 domains

Advantage

End user productivity

Skills

V5 interactive expert to define the V5 design best practices
V5 architect to capture this best practice using BKT

Dev. cost

LOW

The main part of the job is to define the Design Best Practice to be automated

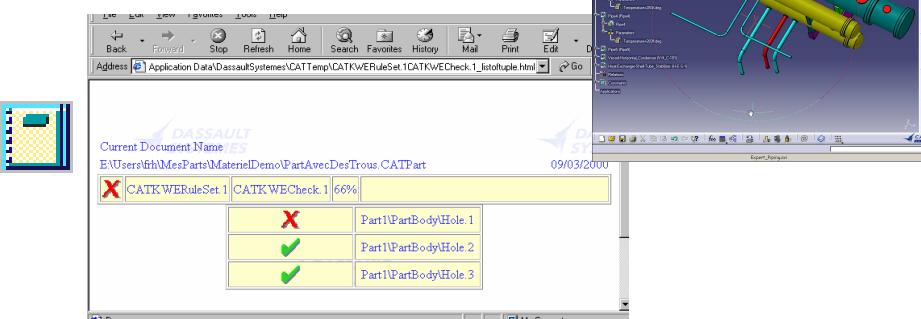
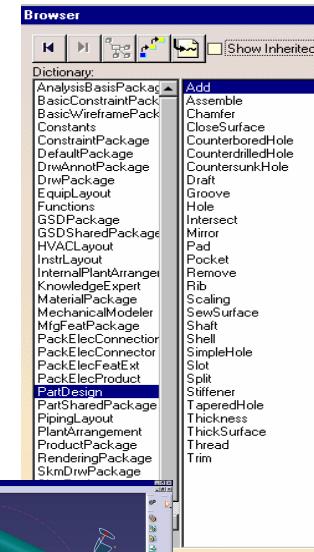
V5 Licenses

BKT to define the customer application
BK2 to use the customer application

Student Notes:

Knowledge Expert

- ❖ Ensure corporate standards compliance and certify design
- ❖ Apply on:
 - ❖ Part & Shape geometry & features,
 - ❖ Drafting, Sheet Metal, Electrical, Digital Mock-Up features
- ❖ Generate reports



Click on movie

[Student Notes:](#)



KWE Characteristics

Scope of Work

All V5 Domains

Advantage

Integration in all the Knowledgeware products.

Skills

V5 administrator to define the corporate rules

Dev. cost

LOW

V5 Licenses

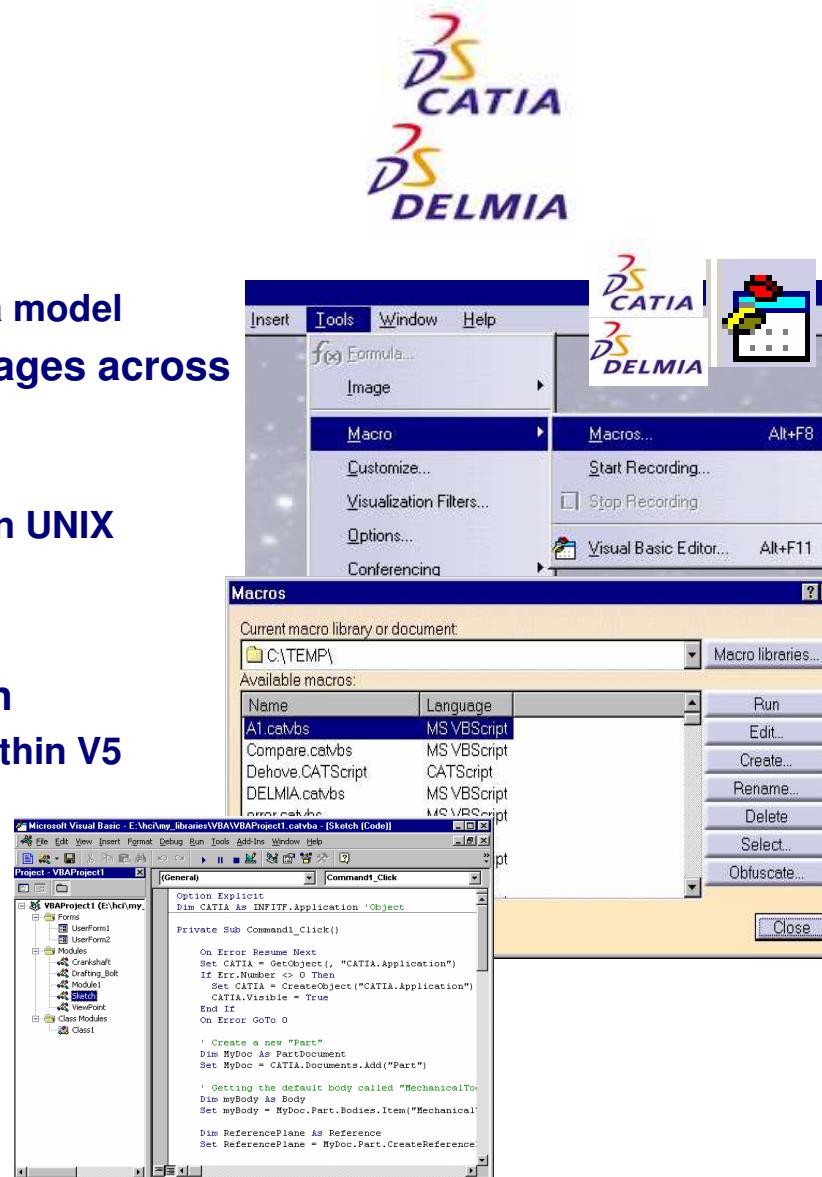
KWE to define the generic rules and checks

KE1 to apply those rules and checks

Student Notes:

VB macros using V5 Automation API

- ❖ **CATIA Automation API:**
 - ◆ end user view on the CATIA data model
- ❖ **Macros relying on standard languages across the platforms**
 - ◆ MS VBScript on Windows
 - ◆ VB Script provided by Winsoft on UNIX
- ❖ **Macro Recording:**
 - ◆ interactive programming
- ❖ **On Windows only, VBA integration**
 - ◆ VB development environment within V5
- ❖ **Selection capabilities**
 - ◆ Feature level
 - ◆ Sub-Element level
 - Face, edge, vertex



[Student Notes:](#)

V5 Automation Capabilities Characteristics



Scope of Work

All V5 domains
In the standard language only simple dialog boxes
“InputBox” and “MsgBox”
More sophisticated dialog boxes in VBA but only on Windows

Advantage

Simple

Skills

VB programmer with V5 knowledge

Dev. cost

MEDIUM

V5 Licenses

No specific license to take advantage of the infrastructure

included in the V5 Kernel

Corresponding Interactive Product Licenses to use the Automation API

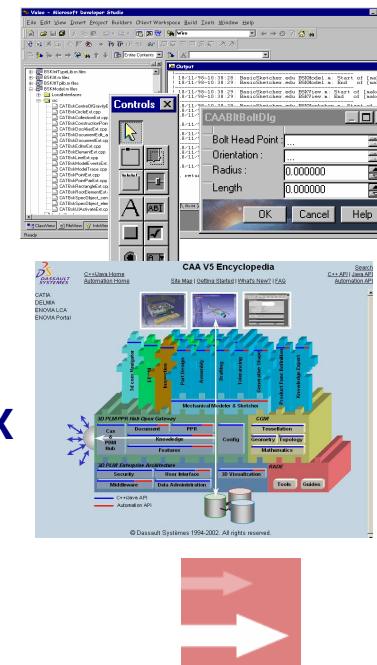
Ex: to use the Part Design Automation API, you need to have a license allowing the usage of the interactive Part Design product

CAA V5



Student Notes:

- ❖ Interactive software development tools built on top of best-in-class de facto standard RADE tools
 - ◆ Ms Visual C++ on Windows
 - ◆ Standard C++ Compilers
 - ◆ CAA V5 Addins
 - Wizards to create workbenches, interactive commands, dialog boxes, ...
- ❖ A huge set of APIs covering all the V5 domains
- ❖ Single source code working on both WINDOWS and UNIX



[Student Notes:](#)



CAA V5 Characteristics

Scope of Work

All V5 domains

Advantage

Powerful
CAA V5 wizards to speed up the development time:
focus on your specificities

Skills

C++ and CAA V5 programmer with V5 knowledge

Dev. cost

MEDIUM

V5 Licenses

CDC or CDV to build the application
V5 interactive license depending on the domain

[Student Notes:](#)

Technological Choices

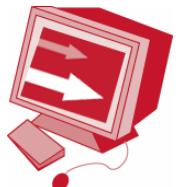


Extend PPR



Using interactive techniques:

- ❖ PKT – Product Knowledge Template
 - ◆ Define new feature types by composing a sequence of existing ones



Using programming techniques:

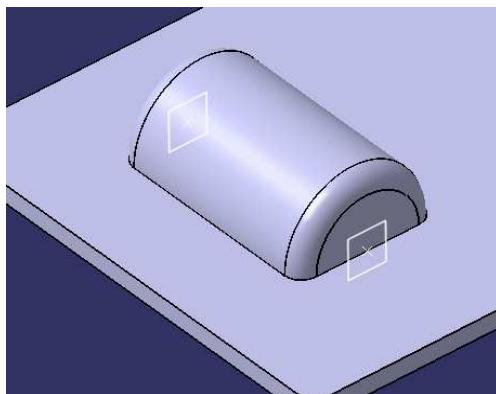
- ❖ CAA
 - ◆ Integrate new geometric objects
 - ◆ Define new features deriving (or not) from DS ones
 - ◆ Add new data and behaviors on an existing feature without overloading standard behaviors

Product Knowledge Template

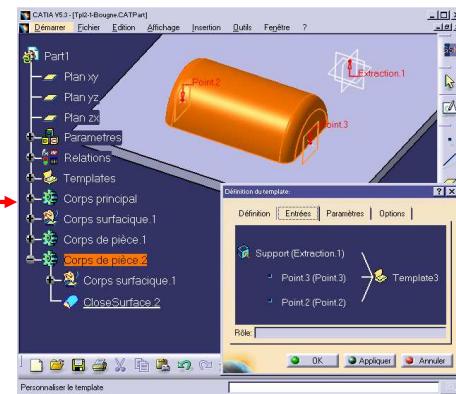


[Student Notes:](#)

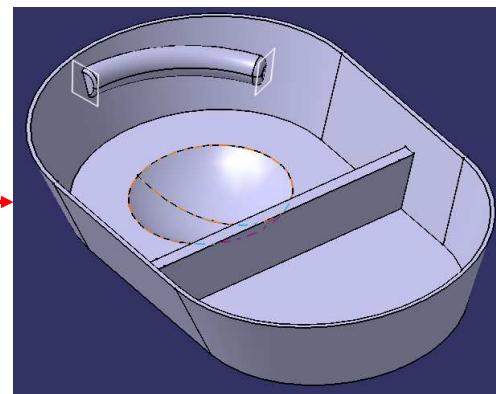
- ❖ Capture and reuse corporate design patterns
- ❖ Create interactively a new feature by aggregating existing ones.
 - ❖ Mechanical features, KW parameters, formulas, constraints
 - ❖ «UDF» to be saved in a CATPart document and referenced in a catalog
 - ❖ Instantiation & adaptation in another context: Morphing



Design in simple context



Define UserFeature



Reuse in Part context

Student Notes:

PKT Characteristics

Scope of Work

Part Design / Generative Shape Design /
Knowledgeware Features only
Product (Assembly Template)
No difficulty to exchange a document including this data



Advantage

Encapsulate a complex sequence of features in a single one
Morphing

Skills

V5 administrator to define the V5 design patterns

Dev. cost

LOW

V5 Licenses

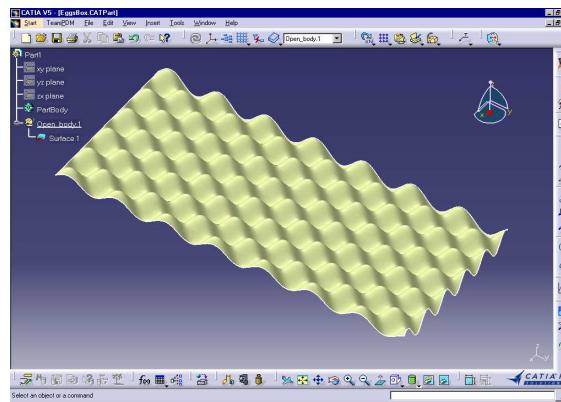
PKT to define the UserFeatures
KT1 to use the UserFeatures

CAA: Foreign Geometric Object



[Student Notes:](#)

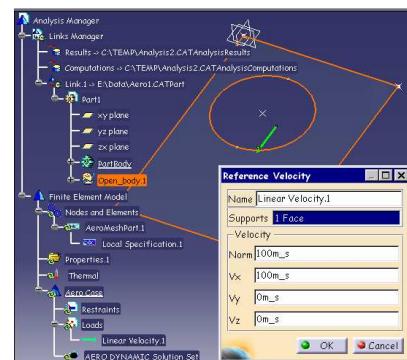
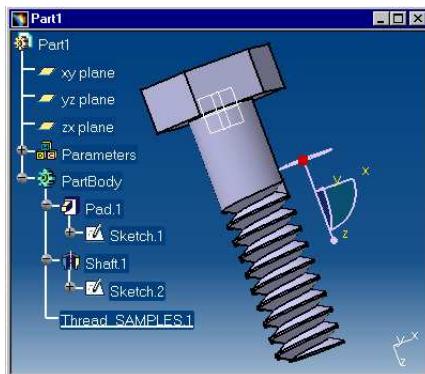
- ❖ **V5 Geometric Modeler provides the most common geometric mathematical representations:**
 - ◆ Canonical
 - ◆ NURBS
 - ◆ ...
- ❖ **Capability to integrate other mathematical representations**
 - ◆ Coons, Gregory Patch, ...
 - ◆ Same constraint as native V5 Geometry:
 - C2 continuity
- ❖ **To be encapsulated in a mechanical feature:**
 - ◆ Datum feature (feature without input)
 - ◆ New mechanical feature



Student Notes:

CAA: DS Feature Derivation

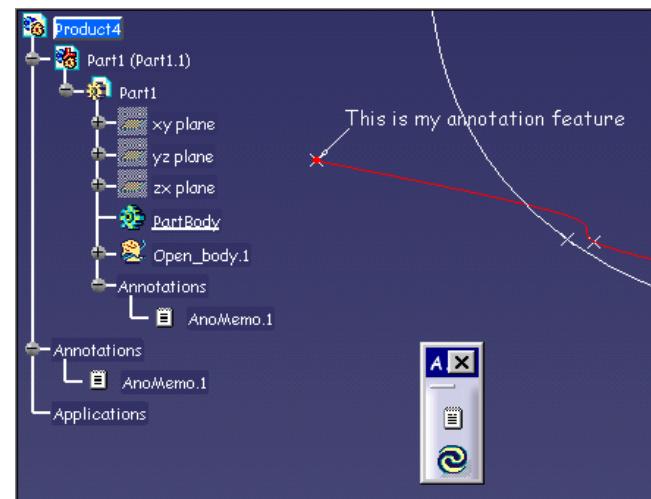
- To integrate your own features in DS applications
 - ◆ Specialize DS features by adding new data
 - ◆ Overload standard behaviors if necessary
 - CATIBuild
 - ◆ Define new behaviors



[Student Notes:](#)

CAA: Feature in an Applicative Container

- Create your own Specifications in a dedicated container
- Define new data structures
- Implement standard DS behaviors or new ones
- Does not extend existing DS Applications
- Same application for different contexts:
 - ◆ CATPart
 - ◆ CATProduct
- Segregate data in several containers



[Student Notes:](#)

CAA: Feature Extension

- Add new data on an existing feature
- No overloading of the base feature behaviors
 - ◆ Except a few through the Provider mechanism
- Define new behaviors
- To make this information accessible, need to activate the application.
- Several applications can extend a feature



CAA



[Student Notes:](#)

Scope of Work

Exchange of a document including this data

Impossible to read it without the application when extending the Geometric Modeler
Needs at least the feature catalog to read the document and the application to edit

Advantage

Reuse: redevelop only what is necessary

Integration

Skills

CAA programmer

Dev. cost

HIGH

V5 Licenses

CDC or CDV to build the application

V5 interactive license depending on the domain

Technological Choices



Student Notes:

Connect

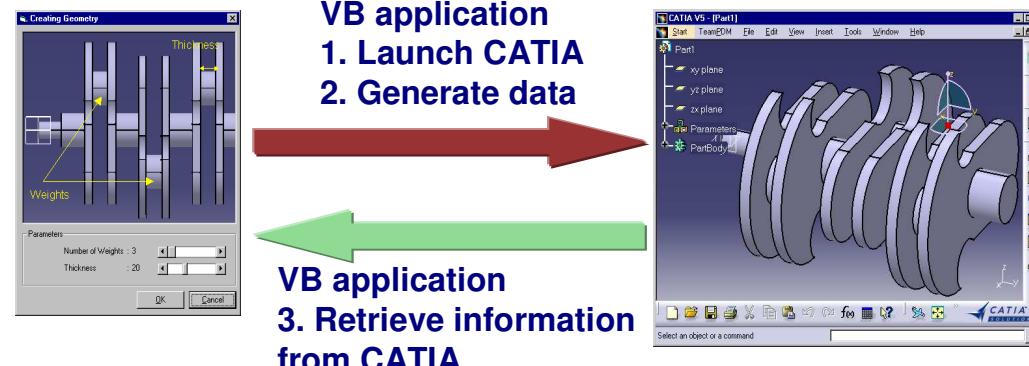
- ❖ Only programming techniques
- ❖ OLE/Automation – Windows only
- ❖ CAA:
 - ◆ xPDM
 - ◆ Backbone

OLE/Automation protocol



Student Notes:

- CATIA/DELMIA is an OLE server.
- Any OLE compliant application can communicate with V5



[Student Notes:](#)

OLE/Automation protocol Characteristics

Scope of Work

Available only on Windows

All V5 documents

Advantage

Standard communication protocol

Simple

Skills

VB programmer

Dev. cost

MEDIUM

V5 Licenses

Corresponding Interactive Product Licenses to use the Automation API

Ex: to use the Part Design Automation API, you need to have a license allowing you to use the interactive product

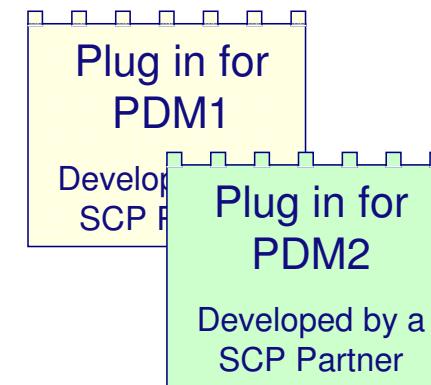
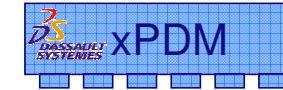
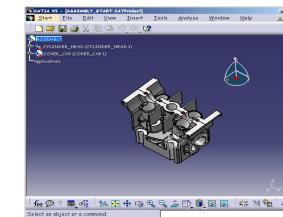
CAA and “xPDM”

CATIA PPR PDM Gateway



[Student Notes:](#)

- ❖ CATIA PPR PDM Gateway enables V5 document management in an external application
- ❖ CATIA objects managed in the PDM System:
 - ❖ CATIA documents
 - CATProduct, CATPart, CATDrawing, ...
 - ❖ Hierarchical Links between CATProducts
 - ❖ Relative Positioning Matrix
 - ❖ Any document link
 - all links displayed in CATIA V5 “Edit links” window giving pointing and pointed elements with document file name



[Student Notes:](#)



CAA and “xPDM” Characteristics

Scope of Work

V5 is the preferred UI to interact with PDM Information.
V5 is the Master to manage the CAD Product Structure.
Concurrent design at the file level
No Relational Design Information (Feature information, Timestamp)
CATProduct to be stored at each node

Advantage

Focus on the specificities of the PDM system you're dealing with

Skills

CAA programmer

Dev. cost

MEDIUM

V5 Licenses

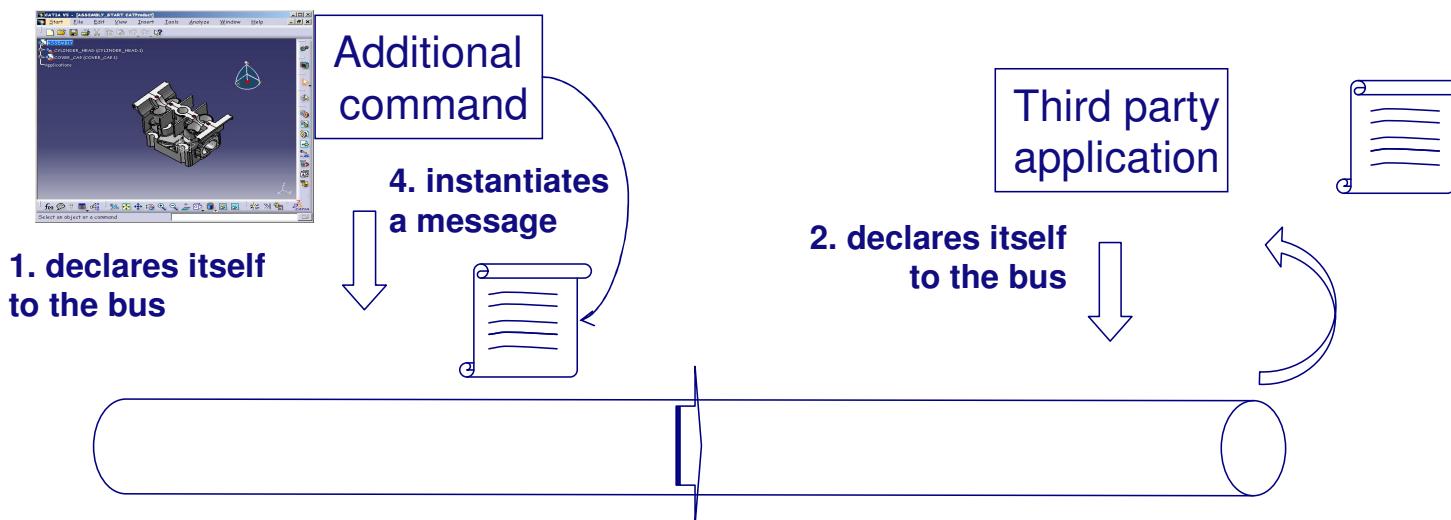
CDC or CDV to build the application
PX1 at run time

CAA: Backbone



[Student Notes:](#)

- ➊ The Backbone is a communication protocol between applications
 - ◆ working on UNIX and WINDOWS
 - ◆ encapsulating the socket technique



[Student Notes:](#)



CAA: Backbone Characteristics

Scope of Work

Communication between processes on the same machine

Advantage

Portability between UNIX and WINDOWS

Simple

Skills

CAA programmer

Dev. cost

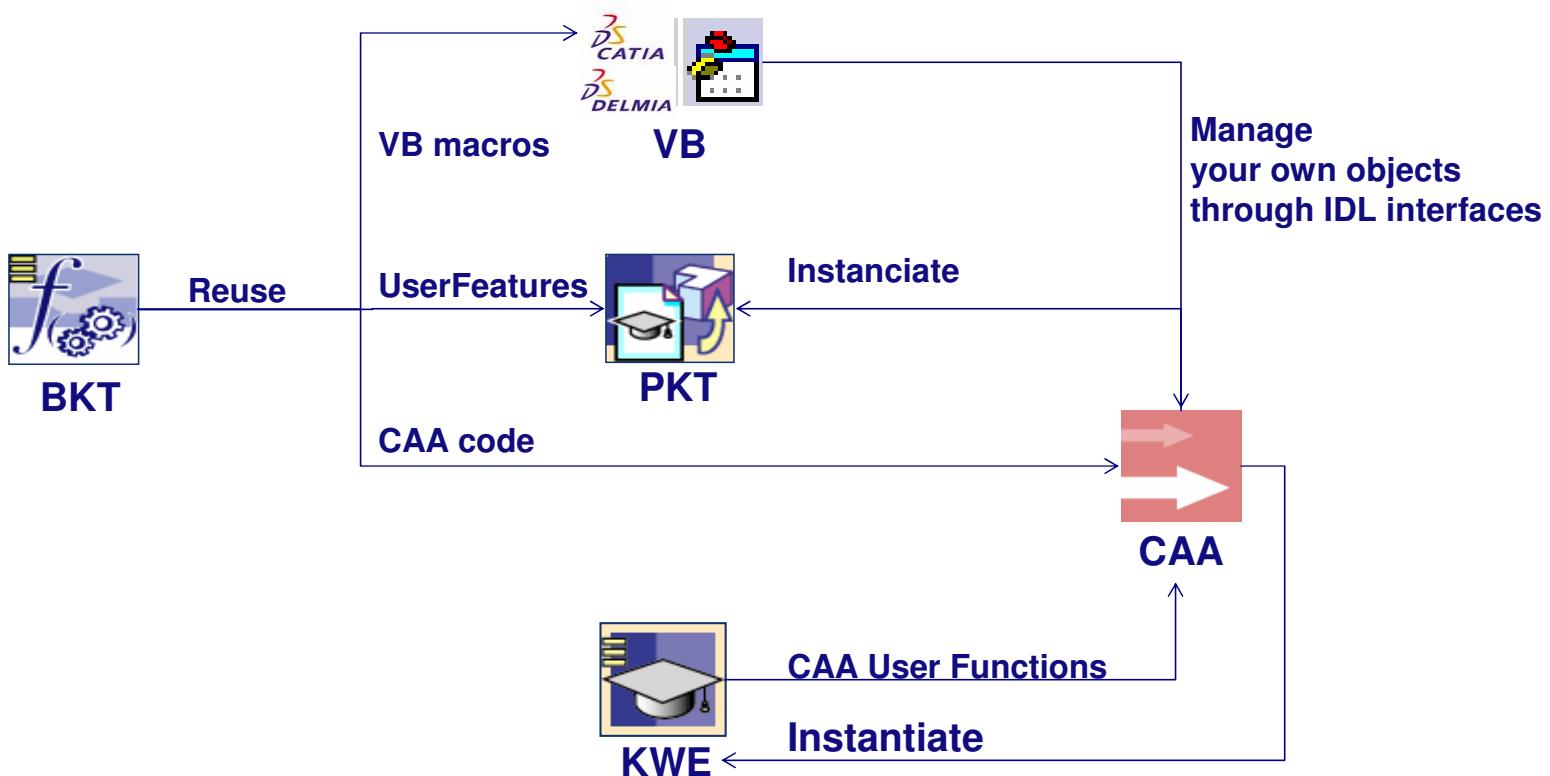
MEDIUM

V5 Licenses

CDC or CDV to build the application

[Student Notes:](#)

Collaboration between all these techniques





Summary

[Student Notes:](#)

	KWE	PKT	V5 Automation Capabilities	BKT	CAA
Create Application	YES	NO	YES	YES	YES
Extend the PPR	NO	YES	NO	NO	YES
Connect Application	NO	NO	YES	NO	YES
Required Skills	V5 Expert	V5 Expert	VB Programmer	V5 Expert	CAA Programmer
Development Cost	LOW	LOW	MEDIUM	LOW	HIGH
License for Build time	KWE	PKT	COM	BKT	CDC or CDV
License for Run time	KE1	KT1	Depends on the used API	BK2	Depends on the used API

To Sum Up ...

- A range of openness tools
 - ◆ Adapted to different targets
 - ◆ From simple to advanced developments
 - ◆ From end-user to professional programmer
- Whatever your needs, you benefit from an appropriate tool.

[Student Notes:](#)

[Student Notes:](#)

CAA V5 Development Environment Basics

You will learn about the specific CAA V5 directory tree structure, the CAA V5 RADE tools plugged in Microsoft Visual C++, and how to find information in the CAA V5 Encyclopedia.

- A Component Architecture
- Workspaces and Frameworks
- Building Tools
- Microsoft Developer Studio Integration
- CAA V5 Encyclopedia

Student Notes:

CAA V5 Development Environment Objectives

- Tools and Methods for an OO programming environment
- Support the V5 Architecture
- Support large teams of developers working concurrently at different sites
- Help making better quality software in a faster way
- Capture and enforce company processes

CAA V5 Characteristics

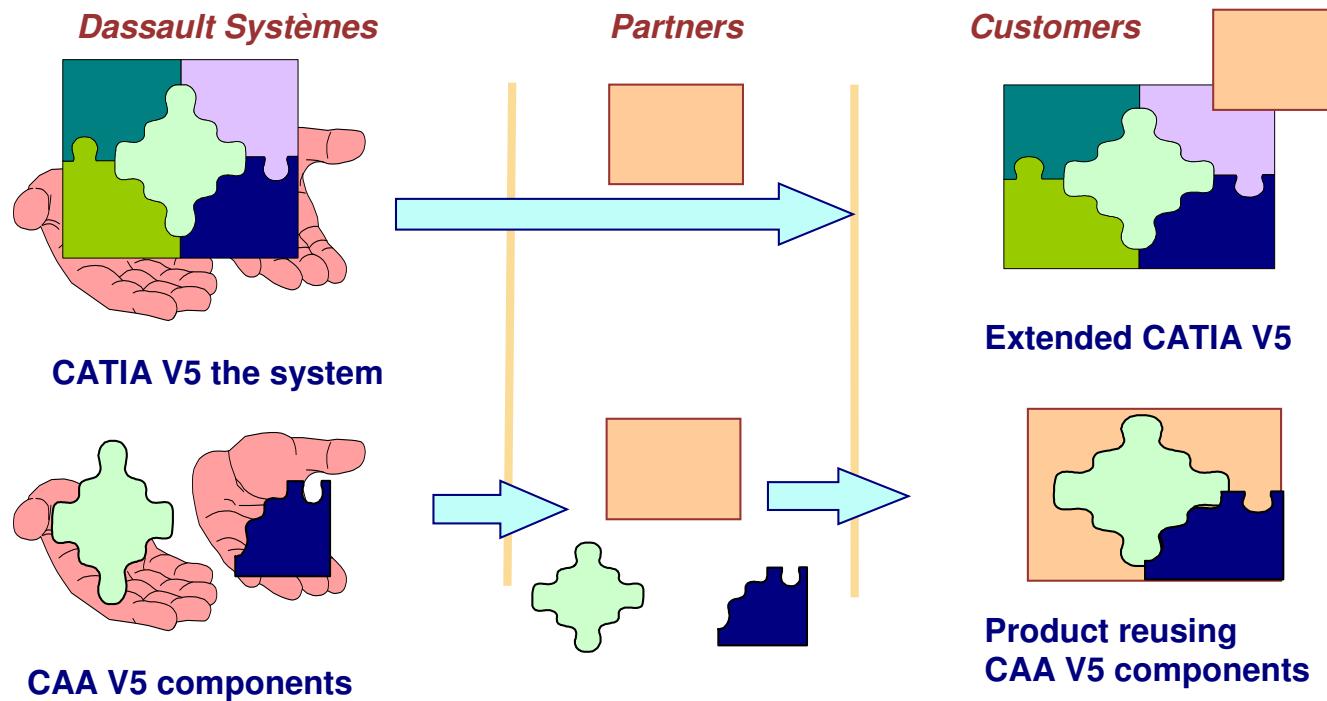
- Common development platform for all the Dassault Systemes product lines
 - ◆ CATIA / ENOVIA / DELMIA
- Code written on top of CAA V5 is the same on WINDOWS and UNIX

[Student Notes:](#)

[Student Notes:](#)

Component Application Architecture

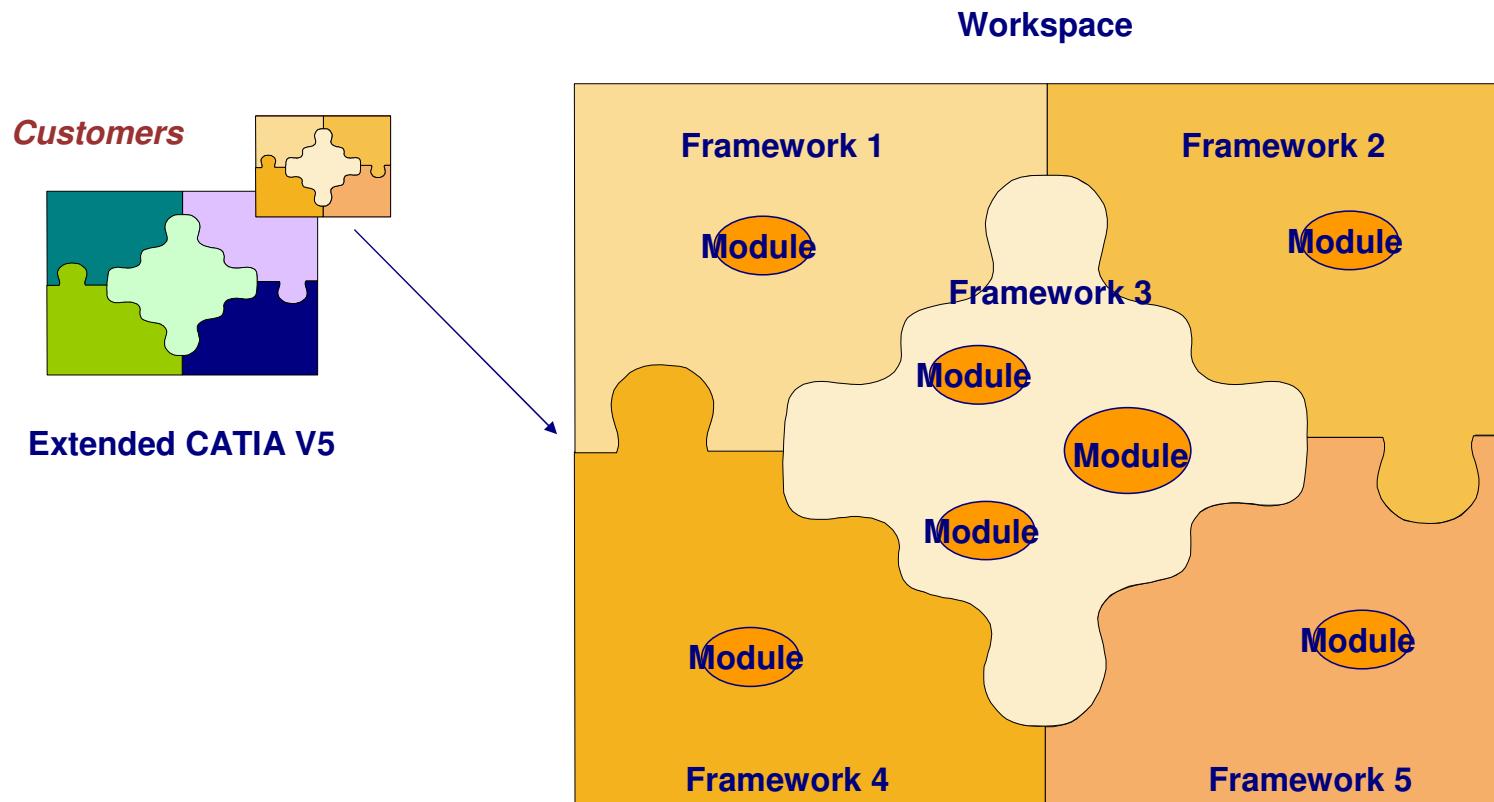
- CAA V5 can be used to implement new products



[Student Notes:](#)

Application Architecture

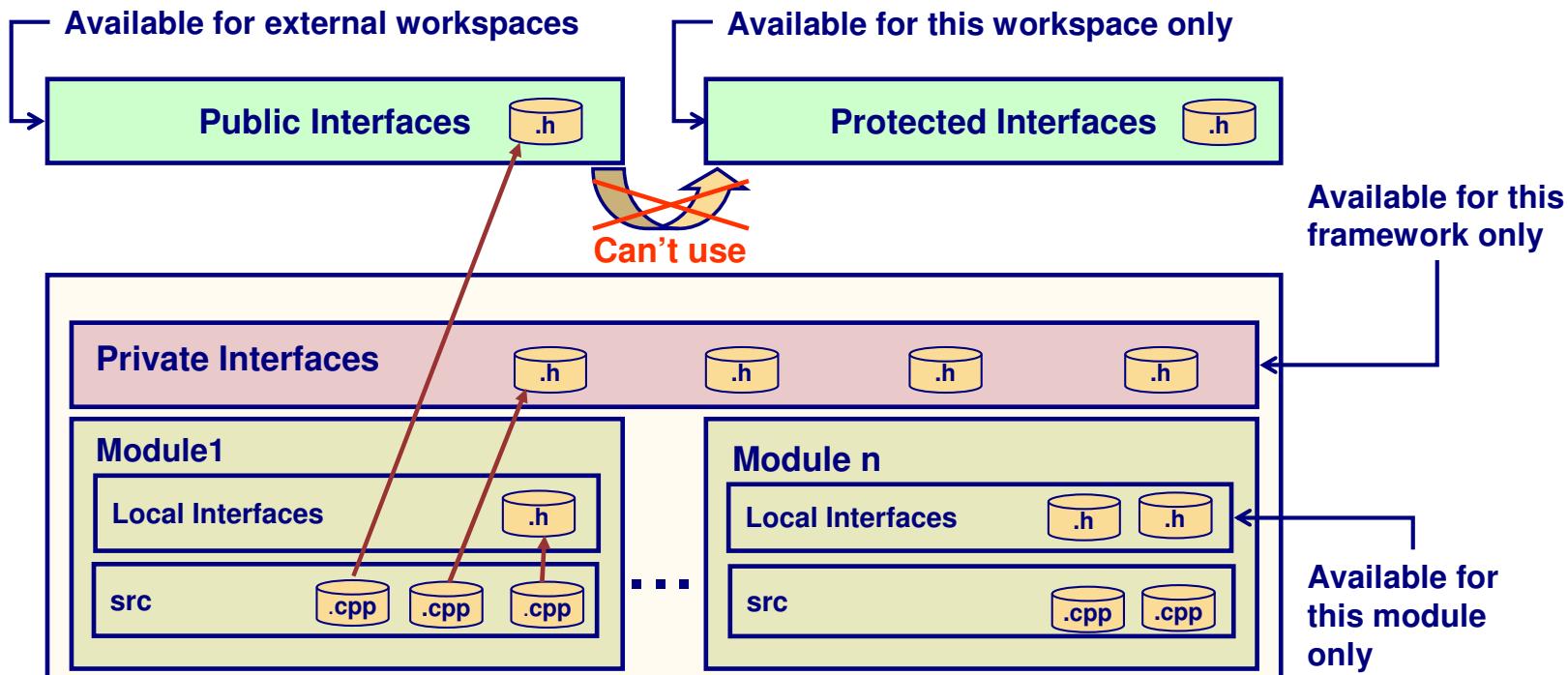
- Every CAA Application is a set of 1 to n components : the frameworks.
- Each framework is composed by 1 to n modules.



Student Notes:

CAA V5 Framework / Module Organization

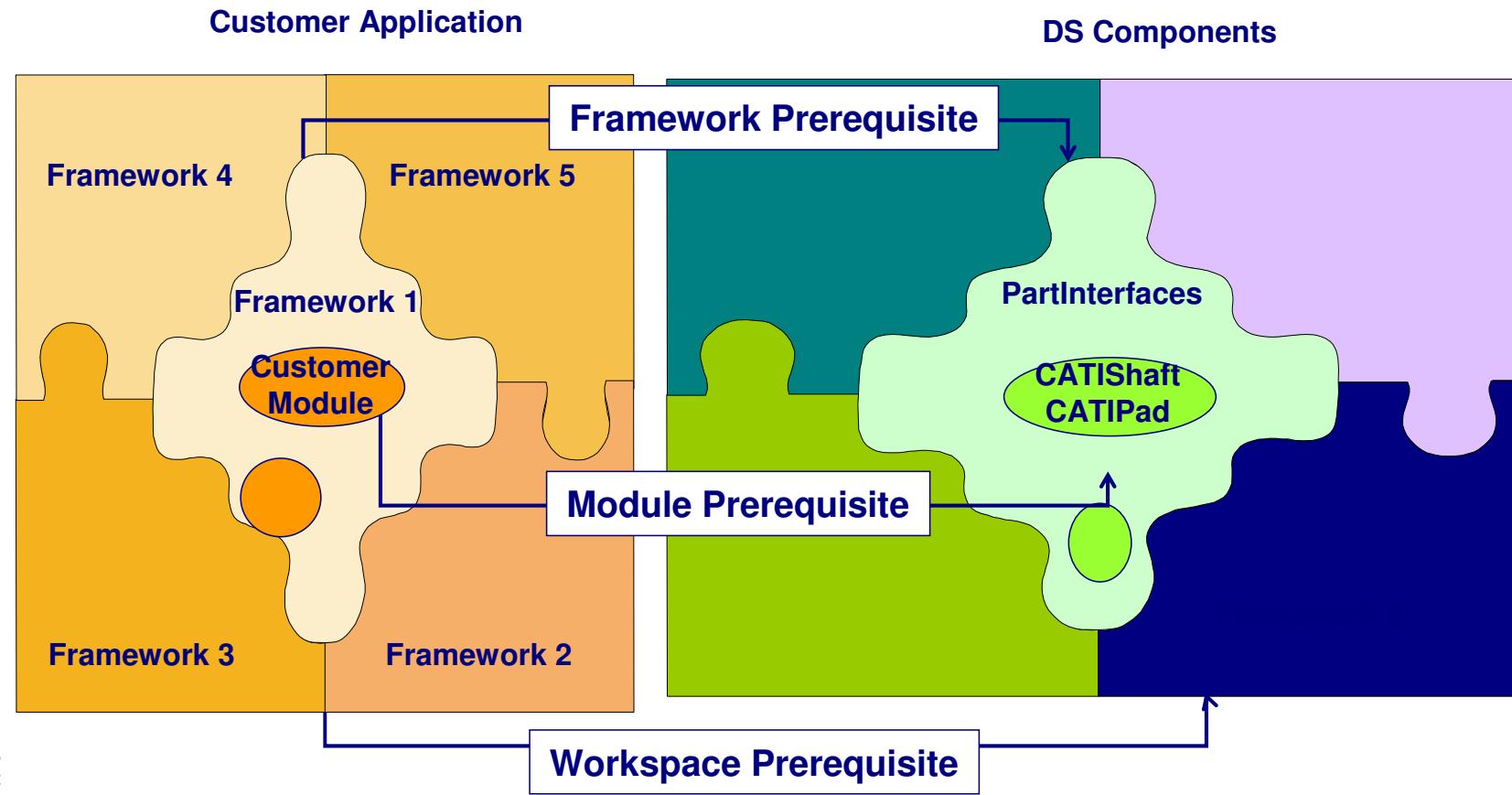
- Group of interoperating objects with built-in capabilities delivered as a complete resource to client applications
- Encapsulation extended at the level of a module and a framework



[Student Notes:](#)

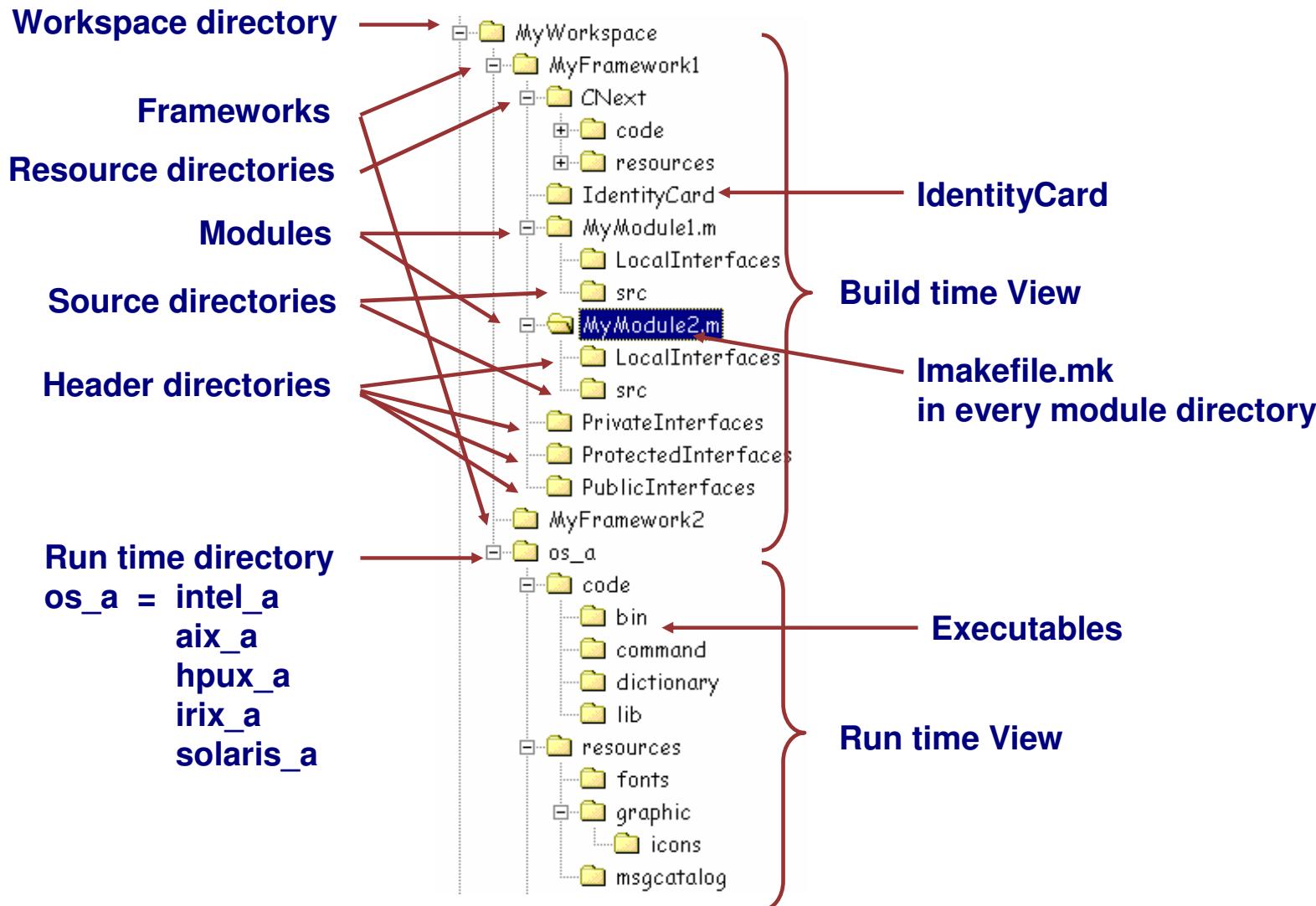
CAA V5 Pre-Requisites

- To access the header file of CATIShaft for build/link time, the customer Application needs the header file complete path : Workspace + framework + module



[Student Notes:](#)

CAA V5 File Tree



Student Notes:

Define your prerequisite workspaces: mkGetPreq

- **mkGetPreq -p PrerequisiteWorkspace1**
- This enables you to define where the prerequisite resources are located
 - ◆ For build time: header files
 - ◆ For run time: shared libraries, resource files, ...
- This command must be launched in a window where the CAA V5 environment has been set and the current directory must be your workspace directory.

Student Notes:

Framework IdentityCard

- The IdentityCard defines the pre-requisite frameworks to build and use a framework.
- One identity card per framework.
- If there is no pre-requisite framework, define an empty IdentityCard.
- This header file is used by our building tool to limit the header file search to the corresponding Interfaces directories of the pre-requisite frameworks.

IdentityCard.h

This framework uses only headers defined in the PublicInterfaces or ProtectedInterfaces directories of the *System* and *ObjectModelerBase* frameworks.

```
AddPrereqComponent("System", Public);  
AddPrereqComponent("ObjectModelerBase", Public);
```

[Student Notes:](#)

mkmk: The Imakefile.mk

- The Imakefile.mk file must be defined for every module.

```
Imakefile.mk
BUILT_OBJECT_TYPE=SHARED LIBRARY           Define the module type
Define the build options common to all the OS
OS = COMMON
WIZARD_LINK_MODULES = \
JS0GROUP JS0FM  CATApplicationFrame          Specific keyword used by the wizards
                                               The continuation character is "\"
LINK_WITH = $(WIZARD_LINK_MODULES) \
CATDialogEngine                            Defines the shared libraries that
                                              resolve the symbols you use
OS = AIX
SYS_INCPATH =                               Define the build options specific to a given OS if necessary
SYS_LIBS = -IXm -IXt -IXmu -IX11 -Im
SYS_LIBPATH = -L/usr/lpp/X11/lib/R5/Motif1.2 -L/usr/lpp/X11/Motif1.2/lib
...
```

[Student Notes:](#)

Manage the CAA V5 Tool Level: TCK

- The Tool Configuration Key manages several levels of the CAA V5 RADE tools.
- To set up the tck environment:
 - ◆ `tck_init`
- To list the different levels available
 - ◆ `tck_list`
- To set up a specific tool level
 - ◆ `tck_profile LevelNameYouWantToUse`

Student Notes:

Build your executables: mkmk

- A unique DS tool built on top of the standard compilers that works in the same way on UNIX and Windows:
 - ◆ Compile Fortran, C, C++, IDL, ...
 - ◆ Link-edit

[Student Notes:](#)

About mkmk

- To access the mkmk help on line, use mkmk -h.
- Use the update (-u) option when:
 - ◆ modifying the dependencies (an include file added or suppressed)
 - ◆ adding or removing a file (.h and .cpp).
 - ◆ modifying the IdentityCard.h and/or the lmakefile.mk
- In other cases, do not use the update option. mkmk will reuse some intermediate files generated before like:
 - ◆ Objects
 - ◆ ImportedInterfaces
 - ◆ various
- Its behavior depends on the current directory:
 - ◆ Your workspace directory is the current directory
 - mkmk –ug to force all the modules to be rebuilt with the debug option.
 - mkmk –a to rebuild only what needs to be rebuilt
 - ◆ A module directory is the current directory:
 - mkmk –ug to force the corresponding module to be rebuilt with the debug option.
 - mkmk to rebuild only if necessary

Build with external libraries

[Student Notes:](#)

Imakefile.mk

OS = Windows_NT

Link with external libraries

LOCAL_LDFLAGS = /LIBPATH:"E:\DirectoryWhereTheLibrariesAreStored"

Name of the libraries

SYS_LIBS = LibraryName.lib

Link with include files

LOCAL_CCFLAGS = /I"E:\DirectoryWhereTheIncludeFilesAreStored"

On WINDOWS

OS = AIX

Link with external libraries

LOCAL_LDFLAGS = -L/MachineName/DirectoryWhereTheLibrariesAreStored

Name of the libraries

SYS_LIBS = LibraryName

Link with include files

LOCAL_CCFLAGS = -I/MachineName/DirectoryWhereTheIncludeFilesAreStored

On UNIX

Student Notes:

ExportedByModuleName Preprocessor Variables

- A Windows mechanism imposes that shared libraries declare explicitly what they import and export.
- To manage this, we define some pre-processor variables in a single header file named as the module.

MyClass.h

```
#include "MyModule.h"
class ExportedByMyModule MyClass
{
...
}
```

MyModule.h

```
#ifdef __WINDOWS_SOURCE
#ifndef __MyModule
#define ExportedByMyModule __declspec(dllexport)
#else
#define ExportedByMyModule __declspec(dllimport)
#endif
#endif
```

Variable defined by
mkmk on Windows

Variable defined by
mkmk
when building MyModule

Student Notes:

Run with external libraries

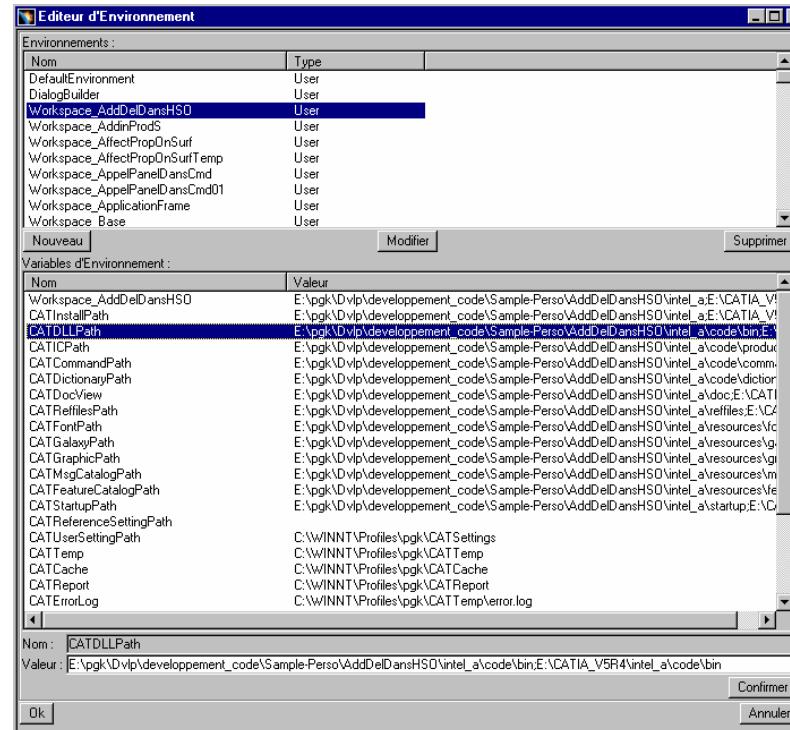
Modify environment

Windows

- Start + Programs + CATIA + Tools + CATIA Environment Editor
E:\CATIA\intel_a\code\bin\CATIAENV.exe

UNIX

- .. / B14 / OS / code / command / catstart – run .. CATIAENV



[Student Notes:](#)

Run time tools

● mkCreateRuntimeView

- ◆ copies the application resources (icons, message files, dictionaries, ...) from the Build time directories into the Run time directories.

● mkrun

- ◆ run CATIA V5 or any main executable developed on top of CAA V5
- ◆ mkrun -c MyProgram

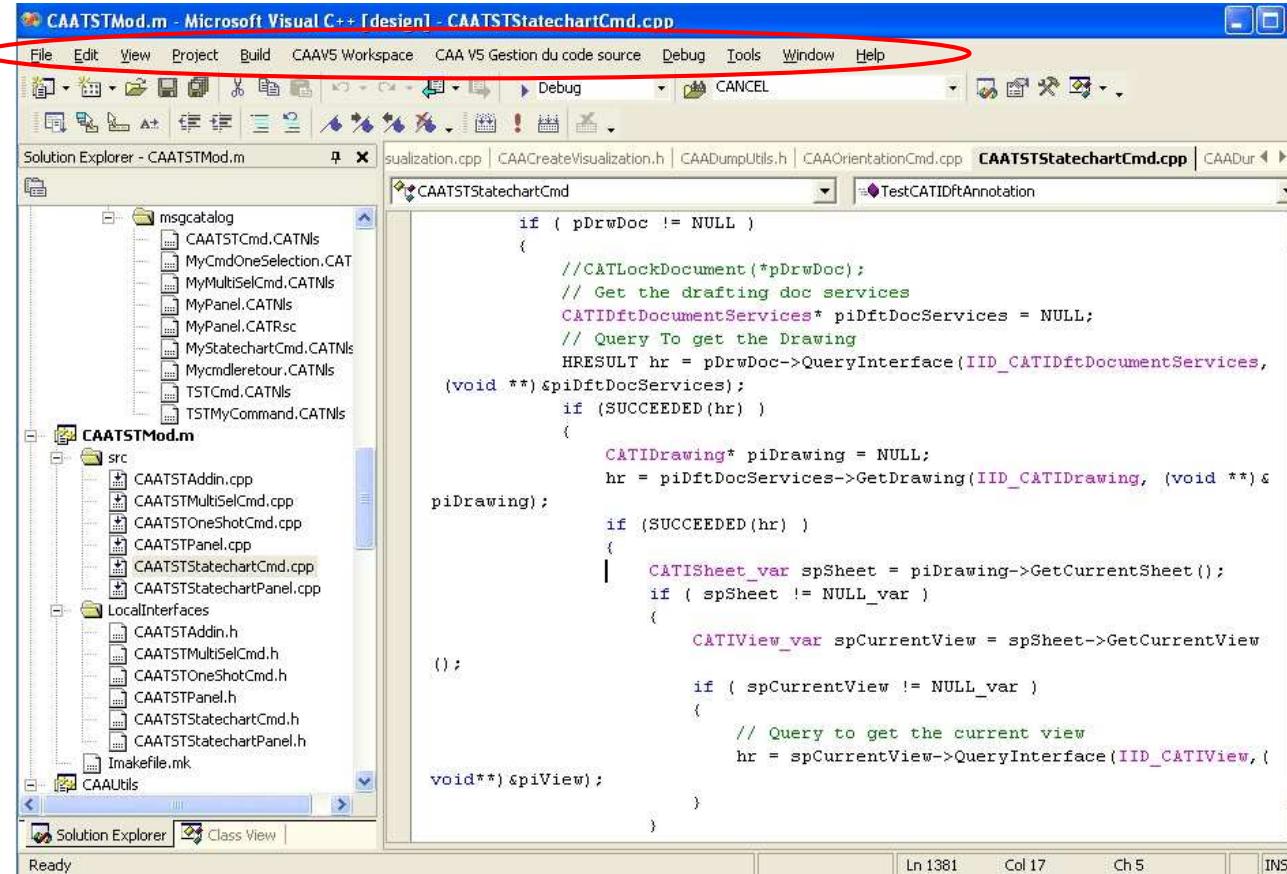
● catstart (available on UNIX, available on WINDOWS from CATIA V5R9)

- ◆ run CATIA V5 or any main executable outside CAA V5 development environment
- ◆ Main Options
 - -env environment_name
 - -direnv environment_directory
 - -run program_name

Student Notes:

Microsoft Developer Studio CAA V5 Add-Ins

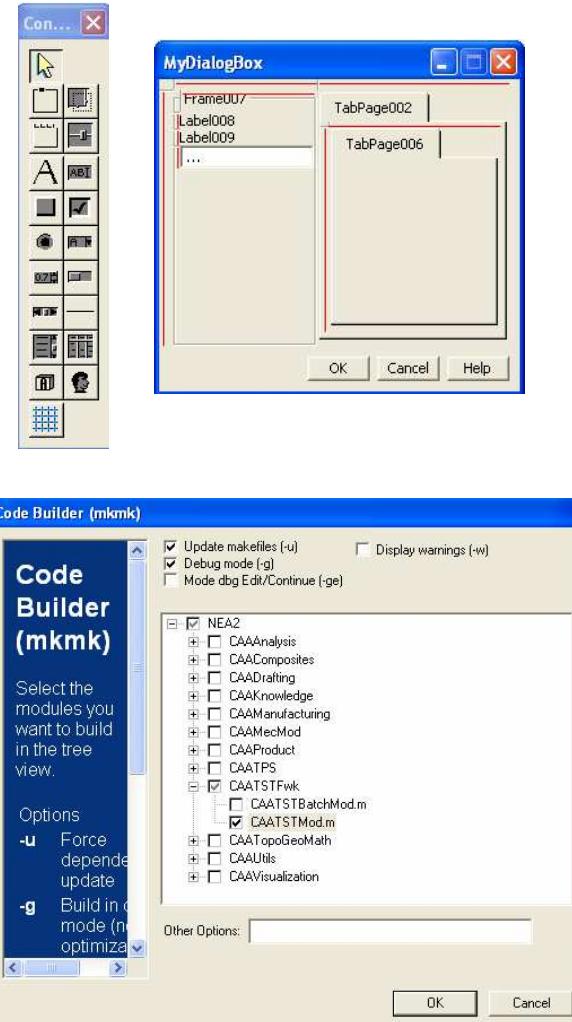
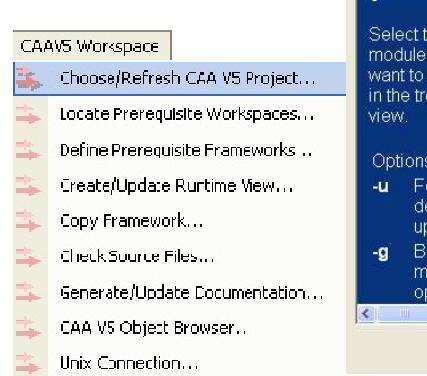
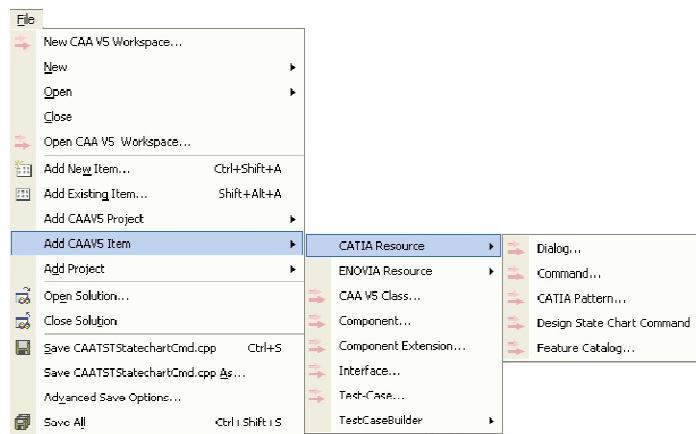
- All our specific tools have been integrated in Microsoft Developer Studio V6
 - ◆ Must be installed using the Unicode String option.



Student Notes:

CAA V5 wizards in Ms Developer Studio

- Wizards to generate code corresponding to generic tasks:
 - ◆ New CAA V5 Workspace
 - ◆ New Framework
 - ◆ New Module
 - ◆ New Command
 - ◆ New Panel
 - ◆ New Interface
 - ◆ New implementation
 - ◆ ...



Student Notes:

Mapping between commands and MsDev Add-Ins

mkGetPreq



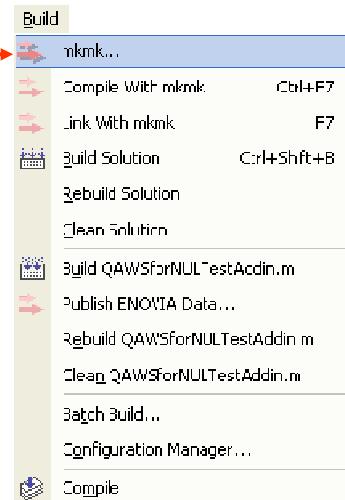
Edit an IdentityCard



mkCreateRuntimeView



mkmk



Student Notes:

MSDev Add-Ins: Hints and Tips (1/3)

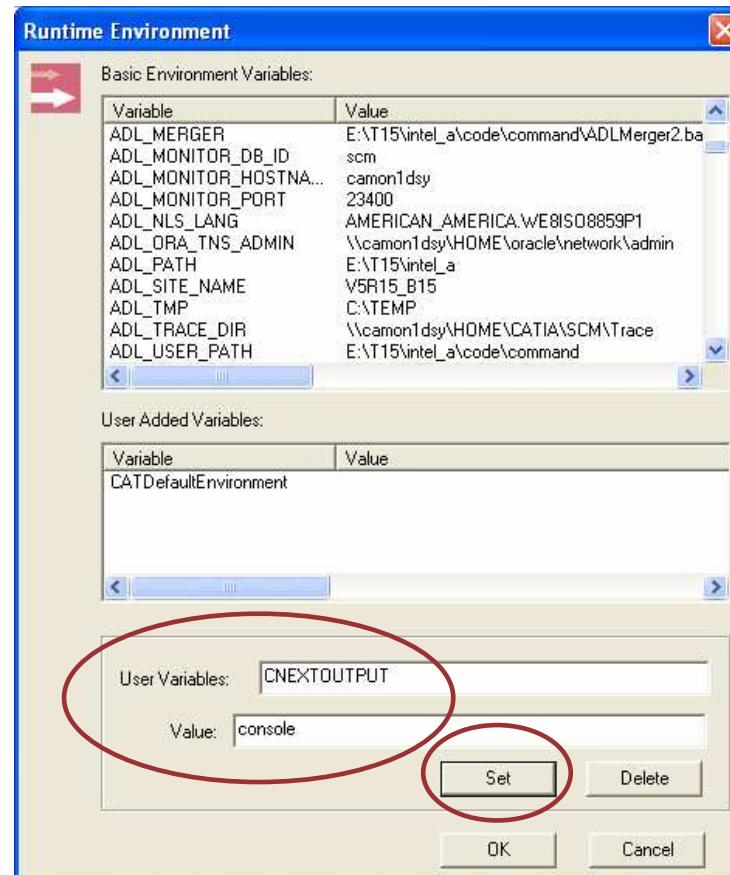
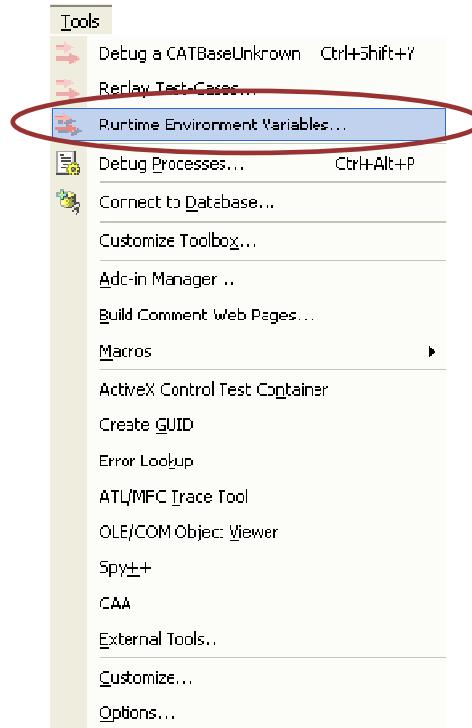
- To see any modification done directly in Windows Explorer (copied, moved or deleted files) in MSDev, use the following command
 - ◆ Project + Choose/Refresh CAA V5 Project...



Student Notes:

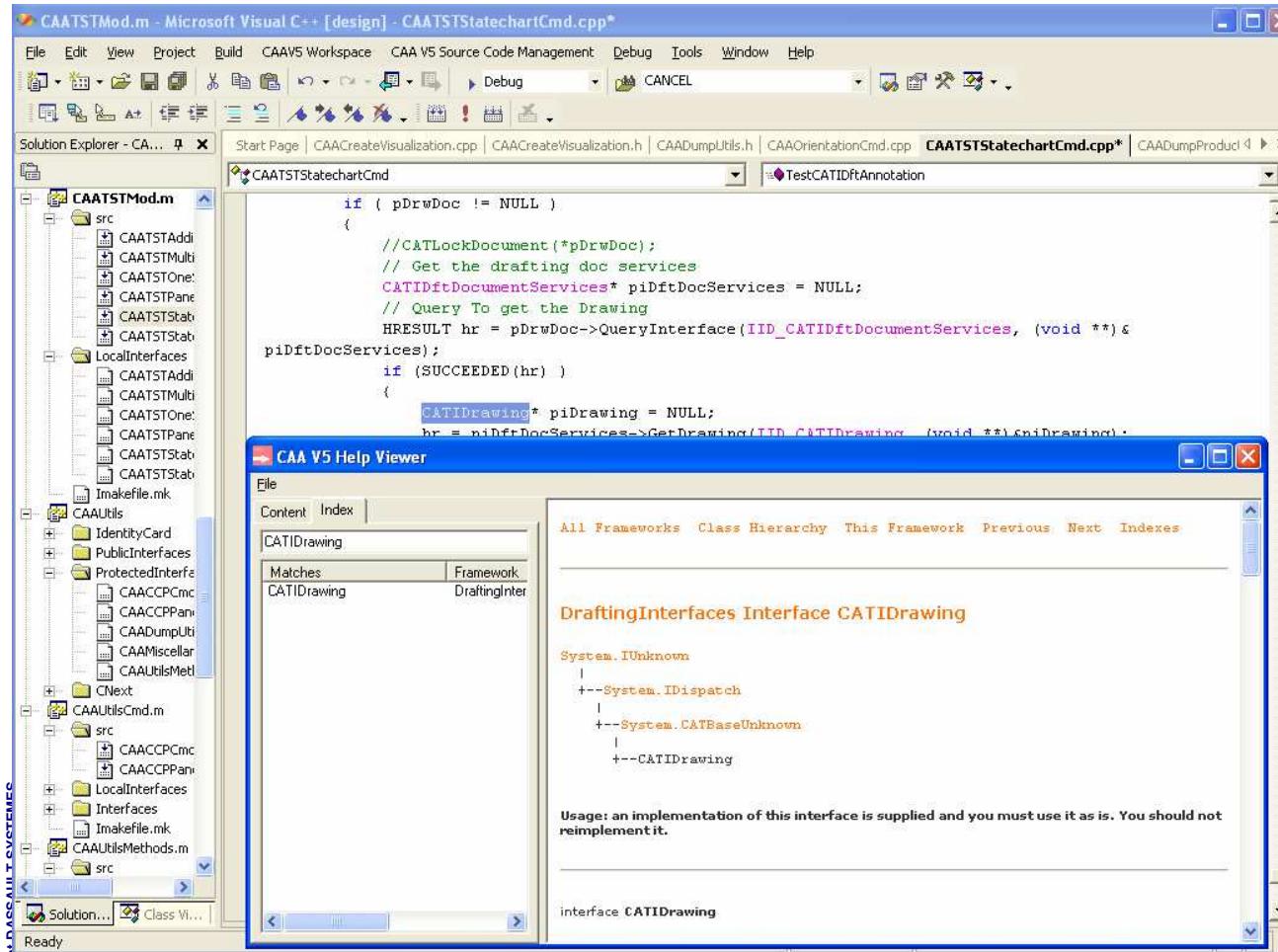
MSDev Add-Ins: Hints and Tips (2/3)

- To see your trace statements, the environment variable CNEXTOUTPUT has to be set to console in Tools+Runtime Environment Variables



Student Notes:

CAA V5 C++ Objects Documentation from Microsoft Visual C++



Position the cursor
on a class name
and key **CTRL+F1**

[Student Notes:](#)

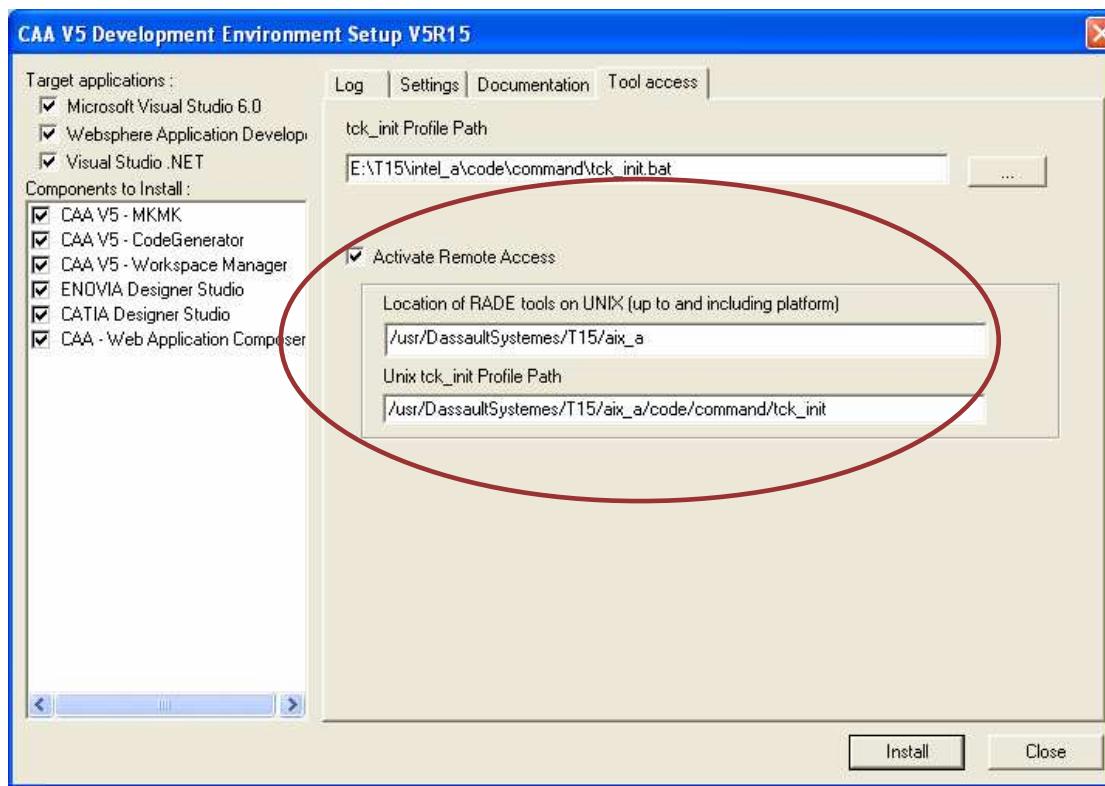
MSDev Add-Ins: Hints and Tips (3/3)

- To rebuild a module and if you don't need the update option, use the keyboard shortcut F7.
- To export a workspace (just the source code) get rid of all the intermediate files generated by mkmk:
 - ◆ go to Tools + Open Command Window and key mkRemoveDo -a
- [Ctrl-Q] to swap between .h and .cpp files
- [Ctrl-T] to open the .h file corresponding to the keyword under the cursor
- [Ctrl-F1] for API documentation
- [Ctrl-Shift-F] to get the full path of the current file

[Student Notes:](#)

Enable porting on UNIX from Visual C++

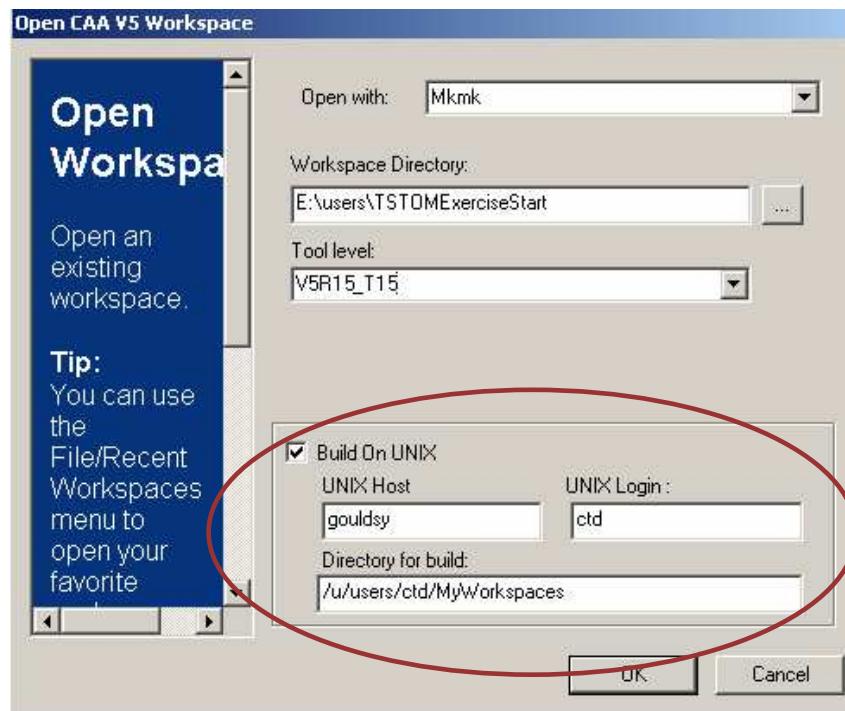
- If you need to port your applications on UNIX, you should run again the CATVBTSetup executable:
 - ◆ C:\Program Files\Dassault Systemes\T07\intel_a\code\bin
- In the Tool access tab page, activate the remote access and inform where the CAA V5 Tools are installed on UNIX



Student Notes:

Activate the Porting on UNIX

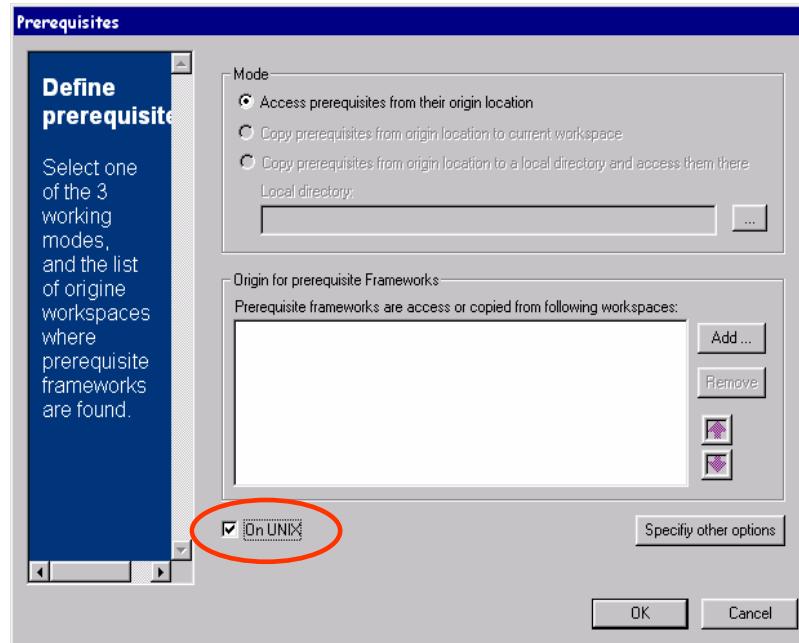
- When opening a workspace, you can ask for building on UNIX by informing Visual C++ on which UNIX machine, with which user and in which directory the operations will be performed.
- Later on whenever a file is generated on WINDOWS, it is copied meanwhile on UNIX



[Student Notes:](#)

Porting on UNIX

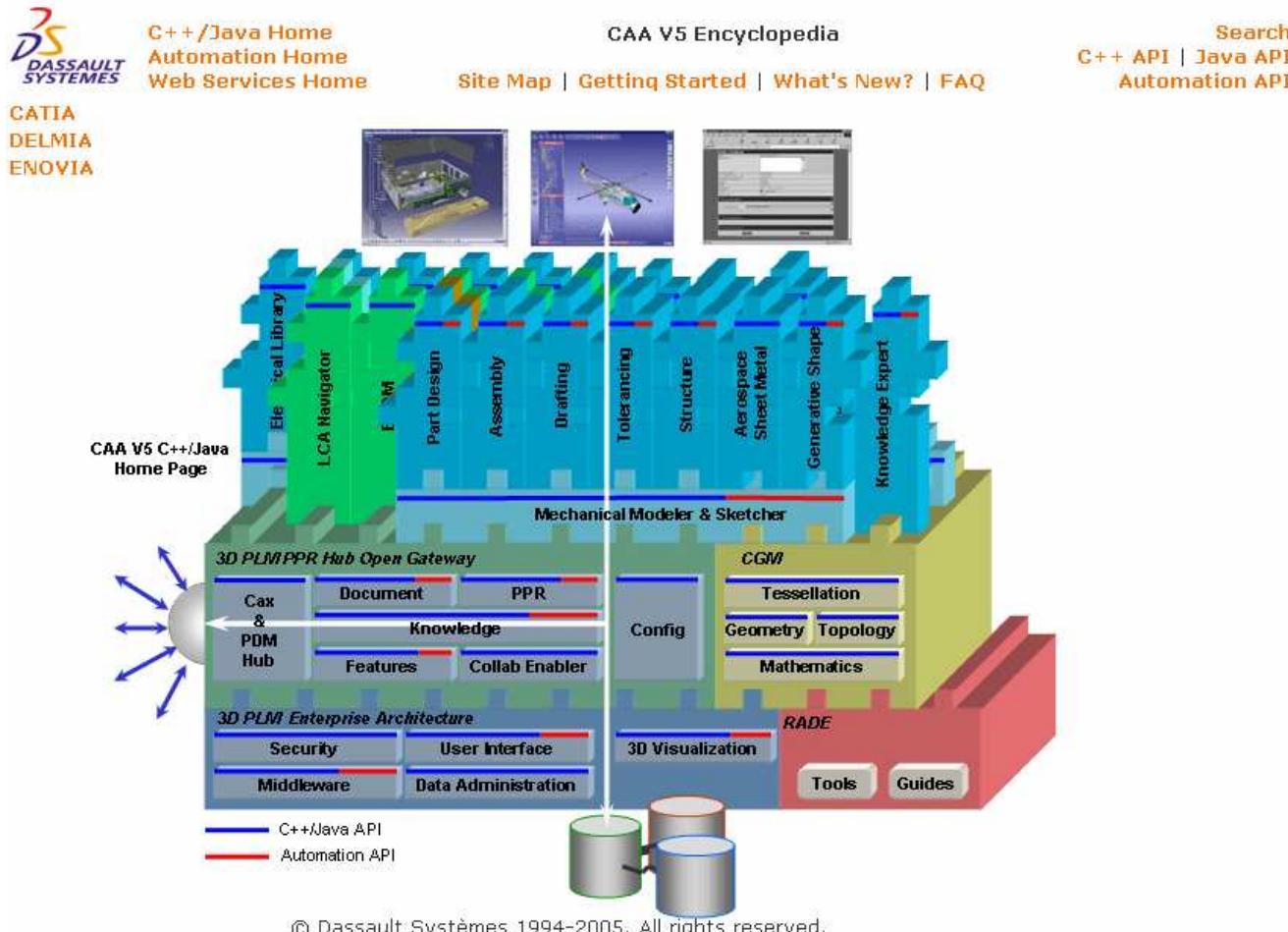
- From Visual C++, then
 - ◆ you can define the pre-requisite workspaces
 - ◆ you can build
 - ◆ you can update the run time view



Student Notes:

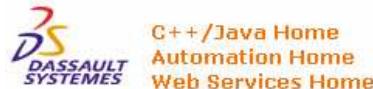
CAA V5 Encyclopedia Home Page

Home page stored in CATIA_directory\CAADoc\Doc\online\CAACenV5Default.htm



Student Notes:

CAA V5 C++ Objects Documentation from a html browser



CAA V5 Encyclopedia
Site Map | Getting Started | What's New? | FAQ



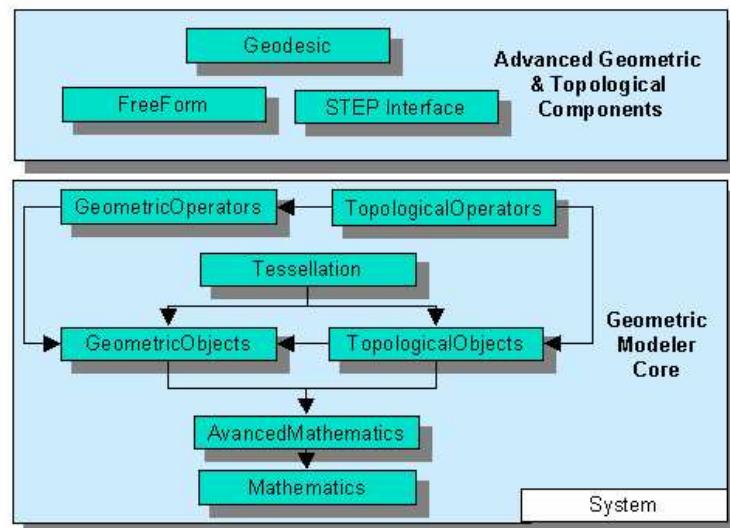
CGM Architecture

The CGM modeler is delivered as a set of specialized components, called frameworks, that are built on top of each other. They can be used stand-alone or integrated with other CAA V5 applications.

[Top]

CGM Frameworks

Each framework provides either the application, or other frameworks with a set of consistent services. The different frameworks cooperate to provide the application with high level services. The hierarchy between the frameworks allows an application which would like to only use a subset of CGM to minimize the size of its prerequisites.



This figure represents the tree of prerequisites and exposed services of the frameworks. For example, GeometricObjects uses as prerequisites some AdvancedMathematics exposed services. Only direct arrows are drawn to simplify the figure. Hence, GeometricObjects also uses Mathematics exposed services.

Presented as a complete geometric and topological package, the CGM modeler can be used to develop powerful applications for building and operating complex geometry.

[Student Notes:](#)

CAA V5 Programmer's Guide (1/3)

- All documentations about a domain
 - ◆ Technical Articles
 - ◆ Use Cases
 - ◆ Quick references

 C++/Java Home Automation Home Web Services Home

CAA V5 Encyclopedia

Site Map | Getting Started | What's New? | FAQ

Search C++ API | Java API Automation API

Geometry

Geometric Modeler

Technical Articles

CGM Overview	A first glance at CATIA Geometric Modeler
The Objects of CATIA Geometric Modeler	An introduction to CGM objects
The Curves of CATIA Geometric Modeler	Properties and detailed descriptions of the CGM curves
The Surfaces of CATIA Geometric Modeler	Properties and detailed descriptions of the CGM surfaces
About NURBS	Concepts and CGM implementation
The Management of Foreign Data	How to put foreign data on CGM objects or to create foreign curves or surfaces
The Clone and Transformation Managers	How to copy and / or move CGM Objects
The Management of Tolerances	Main principles
About the Model Size and Infinite	The validity rules in terms of model dimensions,
The CGM Data Checker	How to check CGM data consistency

[\[Top\]](#)

Use Cases

Foreign Surfaces	How to introduce foreign surfaces
How to Create and Transform Geometry	The basic mechanisms
Using NURBS	Creation of various kinds of NURBS curves
Geometric Operators	The general use explained on an example
Creating a geometric helix	Explains how to create an helix and modify its parameters
Converting Surfaces into NURBS	How to use the Free Form operators to convert surfaces into NURBS.
Converting Curves into NURBS	How to use the Free Form operators to convert curves into NURBS.
Creating an attribute	How to implement and create a streamable attribute.
Reading an attribute	How to search for specific attributes in a geometry.
Creating Explicit Objects	How the life cycle of explicit objects is managed..
Creating Implicit Objects	How the life cycle of implicit objects is managed..

[\[Top\]](#)

Quick Reference

Where to Find What	The index of the resources used in the use cases
Frequently Asked Questions	To help you to answer your questions
GeometricObjects Framework Reference	Interface and class reference for GeometricObjects
GeometricOperators Framework Reference	Interface and class reference for GeometricOperators
FreeFormOperators Framework Reference	Interface and class reference for FreeFormOperators

Student Notes:

CAA V5 Programmer's Guide (2/3)

Technical articles

- ◆ In depth paper
- ◆ Less than 10 pages
- ◆ Hyper linked

The screenshot shows a web page from the CAA V5 Encyclopedia. At the top, there is a navigation bar with links to C++/Java Home, Automation Home, Web Services Home, CAA V5 Encyclopedia, Site Map, Getting Started, What's New?, FAQ, Search, C++ API, Java API, and Automation API. Below the navigation bar, the title "Geometric Modeler Overview" is displayed. A sub-header "Geometric Modeler" is visible. A blue banner at the top of the main content area says "Technical Article". The main content starts with an "Abstract" section, which states: "This article presents the main features of the CGM modeler, in term of objectives, architecture, functionality and openness." It then lists several sections: CGM Objectives, CGM Architecture, CGM Functionality, In Short, and References. Below this, there is a section titled "CGM Objectives" with a detailed description of what the CGM modeler is and how it is used. There is also a note about its design and interoperability.

Abstract

This article presents the main features of the CGM modeler, in term of objectives, architecture, functionality and openness.

- CGM Objectives
- CGM Architecture
 - CGM Frameworks
 - Scalability
 - Openness
 - Integration in the General V5 Offering
- CGM Functionality
 - The Geometric and Topological Modeler
 - Building an Application
- In Short
- References

CGM Objectives

CATIA Geometric Modeler (CGM) is a complete software package for the development of applications with a need for 3D geometric modeling capabilities, and which want to take advantage of the power of the CATIA modeler and its solid, surfacing and wireframe capabilities.

Provided as a set of object oriented programming resources, it is composed of a full set of high level geometric primitives, operations and queries, as well as interfaces dedicated to the integration with other parts of an application such as viewers, dialog monitors and data managers.

Designed in accordance to the major standards and the most recent design patterns, it allows the best interoperability with other software components, a great potential of growth and a possible customization.

[Student Notes:](#)

CAA V5 Programmer's Guide (3/3)

Use Cases

- ◆ CAA V5 Code in Action
- ◆ Step by Step
- ◆ Each step detailed and commented
- ◆ Delivered with fully operational source code
- ◆ Made to be copied/pasted into customer code



[C++/Java Home](#)
[Automation Home](#)
[Web Services Home](#)

[Creating the Geometry Factory](#)

The geometry factory (CATGeoFactory) creates and manages all the CATICGMOBJECT (and the curves and surfaces in particular) [1]. This creation is done by the global function ::CATCreateCGMContainer. Notice that the factory can be defined by reading a NCGM file that was previously stored. In that case, the global function ::CATLoadCGMContainer must be used.

```
CATGeoFactory* piGeomFactory = ::CATCreateCGMContainer() ;  
if (NULL==piGeomFactory) return (1);
```

[CAA V5 Encyclopedia](#)

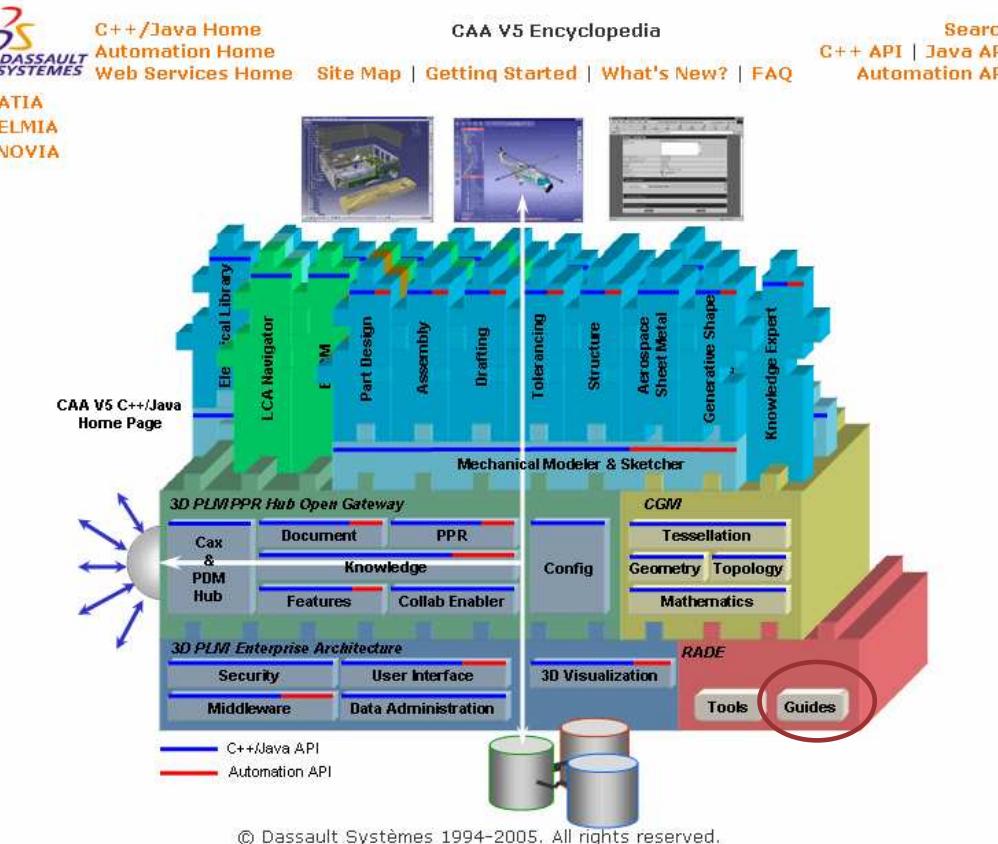
[Site Map](#) | [Getting Started](#) | [What's New?](#) | [FAQ](#)

[Search](#)
[C++ API](#) | [Java API](#)
[Automation API](#)

Student Notes:

CAA V5 Programming Rules

- Programming Rules
- Naming convention
- C++ coding rules
- Java coding rules
- Architecture rules
- User Interface look and feel
- Available in the encyclopedia



[Student Notes:](#)

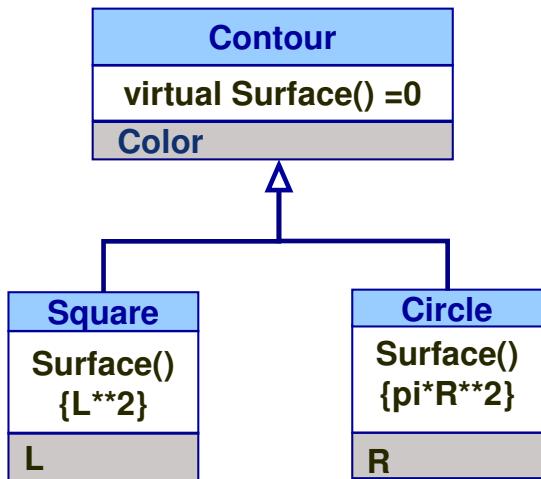
CAA V5 ObjectModeler: Interface / Implementation Design Pattern

In this lesson you will learn the CAA V5 OO infrastructure, the interface handling, shielding any client application from implementation details.

- Why a New Object Modeler?
- Interface / Implementation
- Interface / Implementation Rules
- Interface / Implementation at work
- Interface and Implementation with MsDev wizards
- Interface / Implementation limitation
- Factory
- Coding Rules

[Student Notes:](#)

C++ as a Starting Point

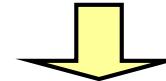


- **Encapsulation**
 - ◆ Hiding implementation details
 - ◆ Changing private part without client impact
- **Inheritance**
 - ◆ Reusing implementation
 - ◆ Extending types
- **Polymorphism**
 - ◆ Genericity on client side
 - ◆ Specialization of provider side

[Student Notes:](#)

What's wrong with C++ ? (1/3)

- Encapsulation could be better
 - ◆ Compilation link between object and its client
 - ◆ even if only a private member changes



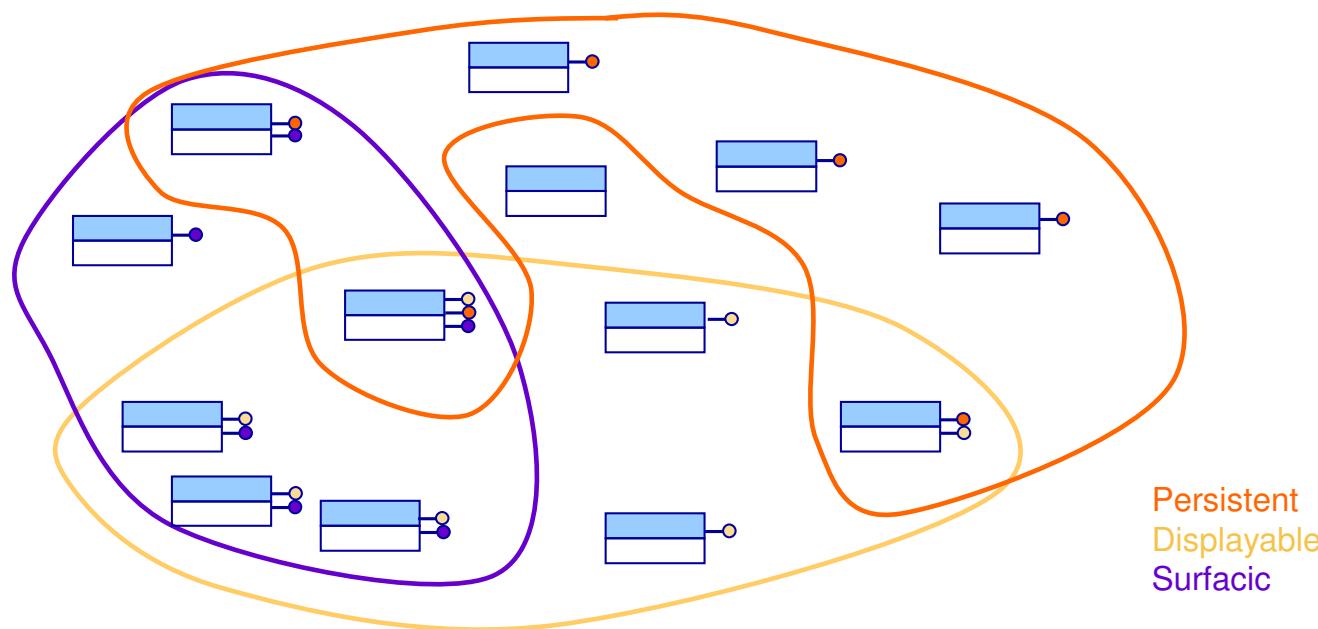
- Not enough run time flexibility
 - ◆ Cannot instantiate classes by name
- Needed for
 - ◆ Adding new types to the system...
 - ◆ ... and let «old» system instantiate them

~~void* myObj = new(iargv[1])~~

[Student Notes:](#)

What's wrong with C++ ? (2/3)

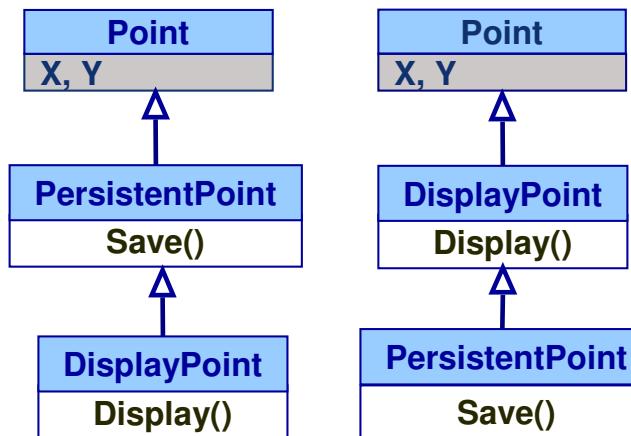
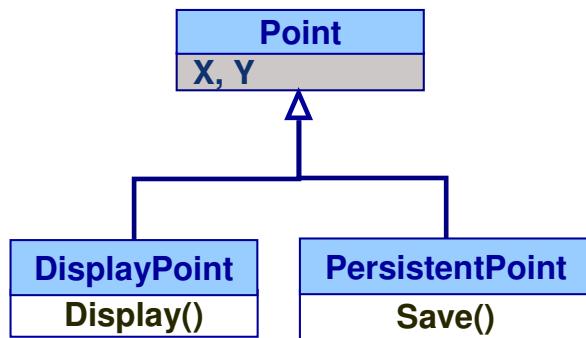
- No support for «schizophrenic» objects
- In large systems, objects have more than one type. These types are needed or useless according to the context.
- Multiple inheritance is not a solution



[Student Notes:](#)

What's wrong with C++ ? (3/3)

- Extension only through inheritance
 - How to organize inheritance trees?
 - What if extensions added later?
 - What about code modularity?



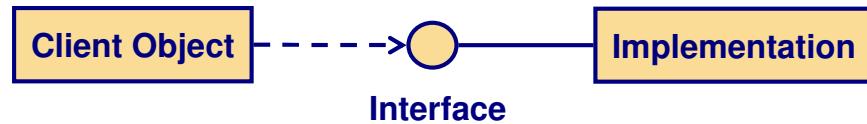
Student Notes:

The Interface / Implementation Pattern

- Client and Implementation separation is achieved by the mean of interfaces.
- Interface shields the client object from the implementation details.



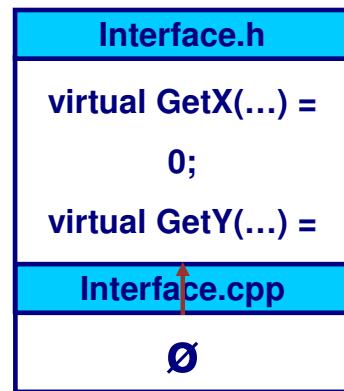
Alternative notation :



Student Notes:

Interface / Implementation Definition (1/2)

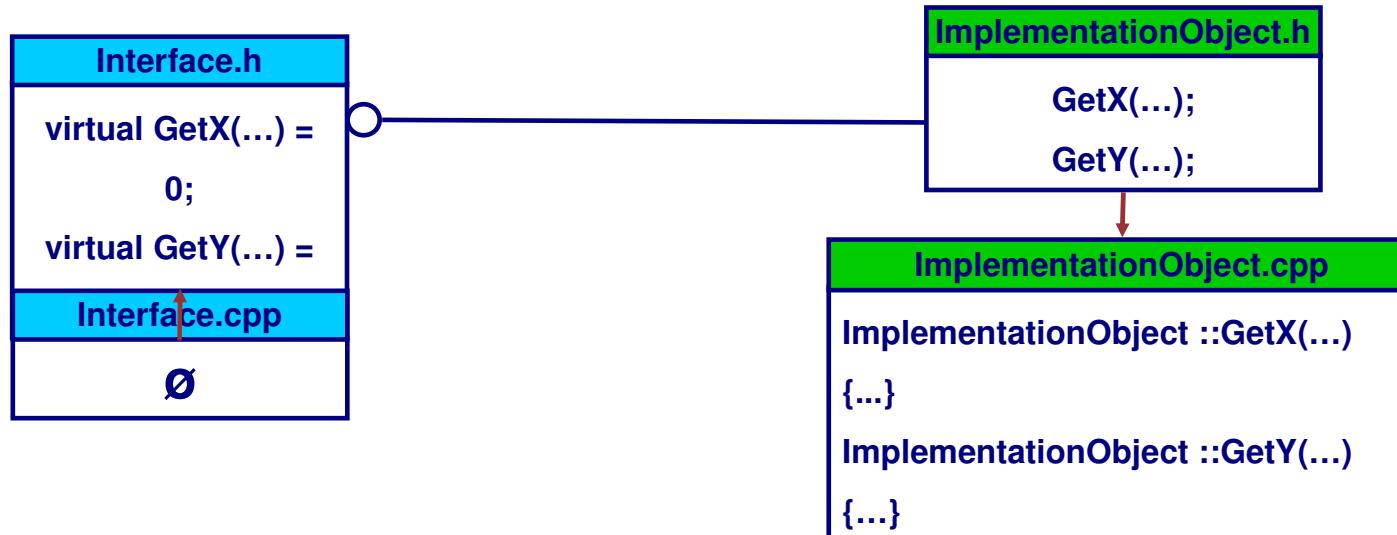
- An interface is an abstract object with a set of pure virtual methods that defines a behavior that objects are subject to support.
- An interface does not know which objects implement it.
- An interface sets a contract between a client and provider's code
 - ◆ Function prototypes
 - ◆ Design intent



[Student Notes:](#)

Interface / Implementation Definition (2/2)

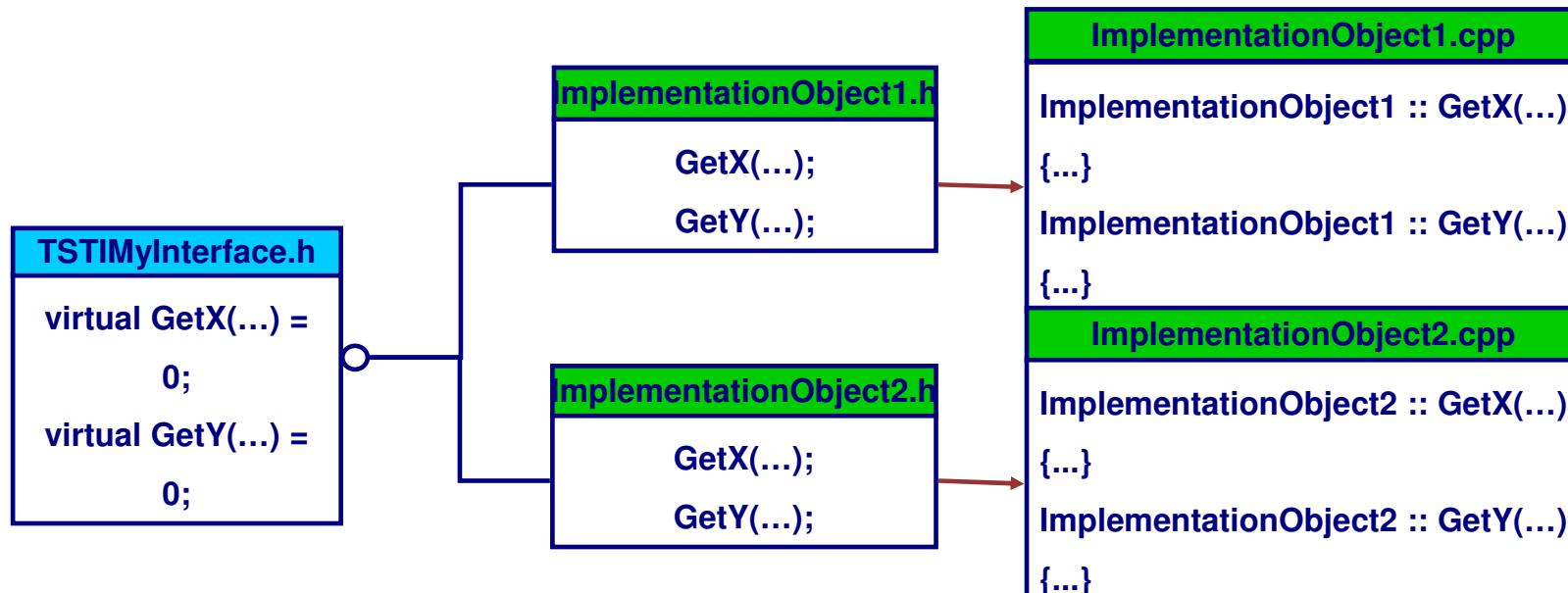
- An implementation is an object that defines a specific way to fulfill the contract set by an interface.
- An implementation has to explicitly state which interface(s) it adheres to.
- An implementation must provide code for all the abstract methods defined in the interfaces it adheres to.
- The client application deals with the implementation only through the interface.



Student Notes:

Interface / Implementation Rules

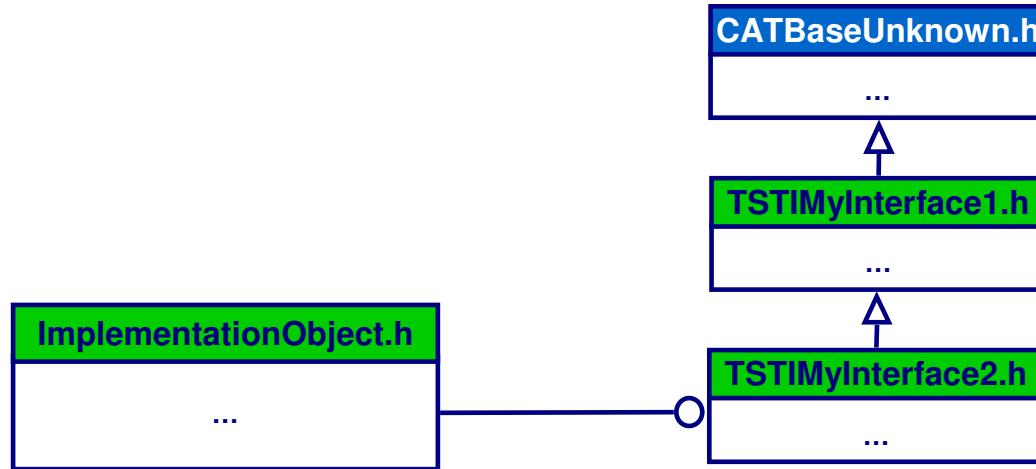
- An interface's name gets always an « I » as the 4th character.
- DS interface names begin by CATI .
- An interface can be implemented by several objects in different ways.



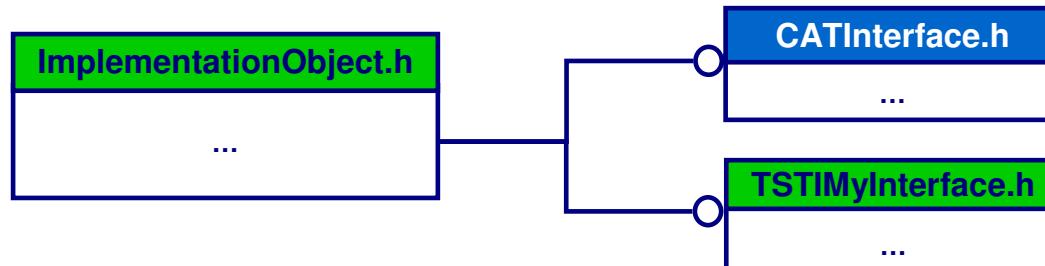
[Student Notes:](#)

Interface / Implementation Inheritance (1/2)

- Interfaces can inherit from each other
 - But at least, the first one must inherit from IUnknown / CATBaseUnknown



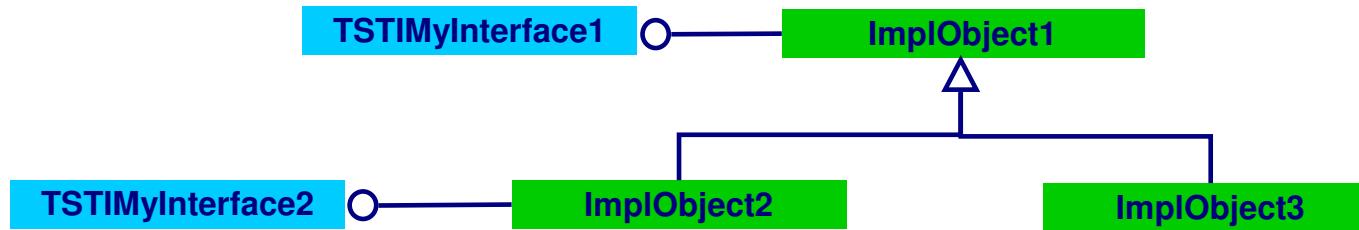
- An implementation may adhere to several interfaces.



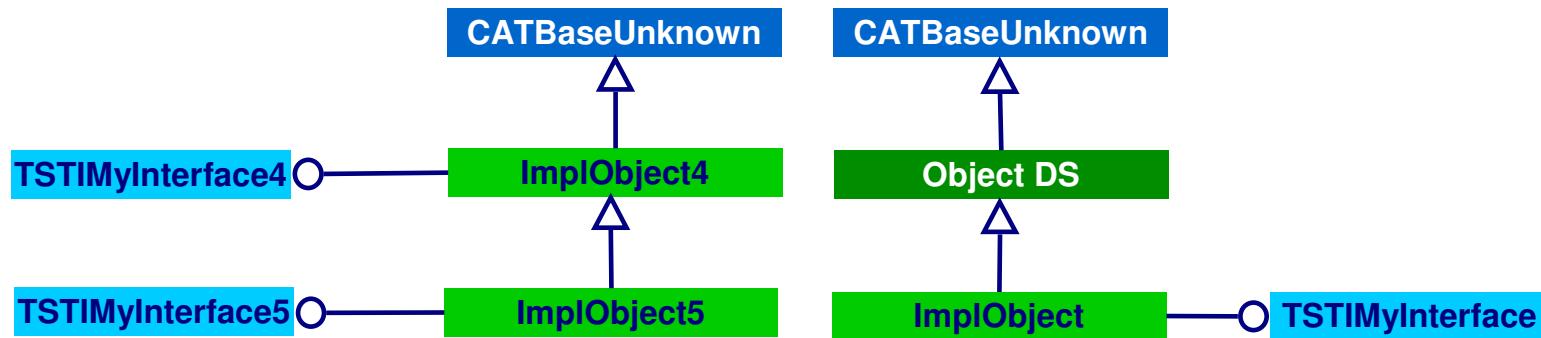
Student Notes:

Interface / Implementation Inheritance (2/2)

- ImplObject2 adheres to TSTIMyInterface2 and to TSTIMyInterface1 by inheritance from ImplObject1, whereas ImplObject3 adheres only to TSTIMyInterface1.



- An implementation can inherit from a C++ class or from another implementation
 - But at least, the first one must inherit from IUnknown / CATBaseUnknown



[Student Notes:](#)

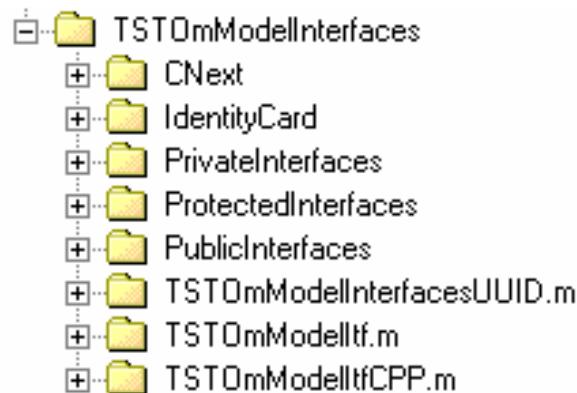
Interface at work : the GUID

- The Global Unique Identifier (GUID) of an Interface is stored in a specific module :
- Module's default name generated by the CAA wizard : MyFrameworkInterfacesUUID.m
- Name of the file : TSTIMyInterface.cpp

TSTIMyInterface.cpp

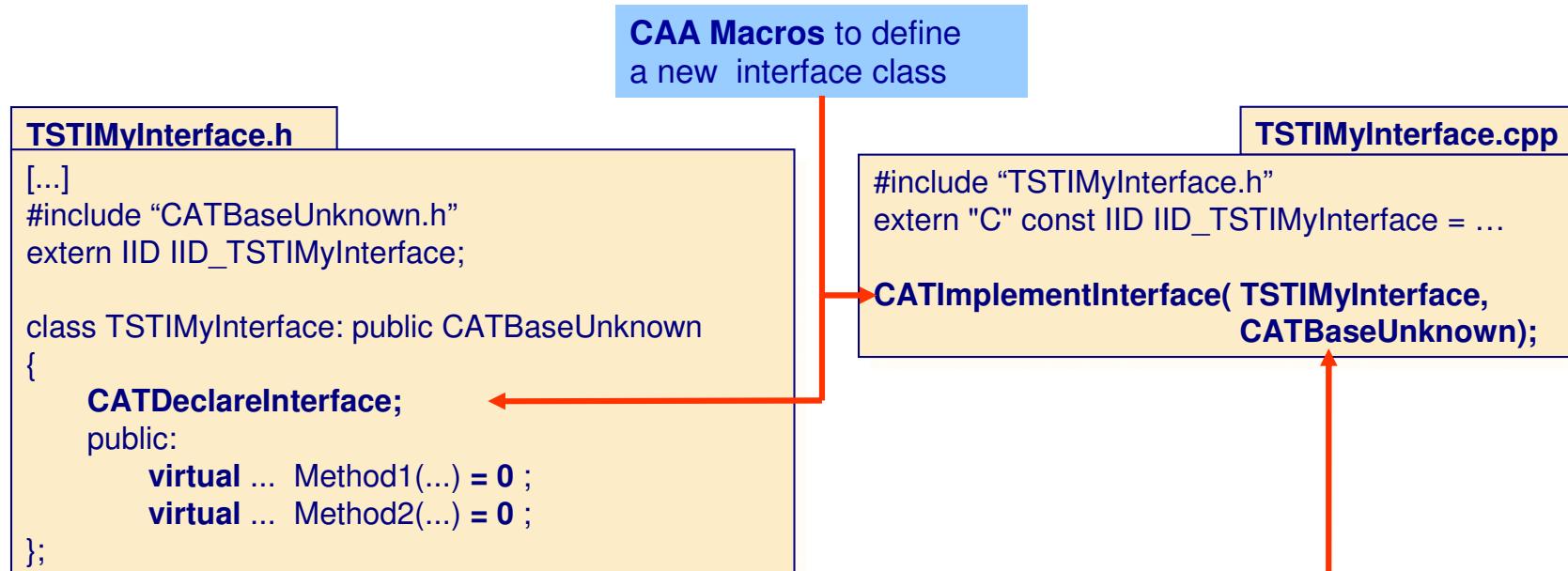
```
#include "IUnknown.h"
extern "C" const IID IID_TSTIMyInterface =
{ 0x32bbc9aa, 0x0254, 0x11d5, { 0x85, 0x08, 0x00, 0x02, 0xb3, 0x06, 0x11, 0x71 } };
```

- The MyFrameworkInterfacesUUID and MyFrameworkInterfacesItf (or MyFrameworkInterfacesItfCPP) modules has to be included in the Imakefile.mk files in other Frameworks for accessing the TSTIMyInterface interface.



[Student Notes:](#)

Interface at work



Student Notes:

Implementation at work (1/2)

ImplObject.h

```
[...]
#include "CATBaseUnknown.h"

class ImplObject: public CATBaseUnknown
{
CATDeclareClass;           ←
public:
    ImplObject() ;
    ~ImplObject() ;

    // TSTIMyInterface adhesion
    ... Method1(...) ;
    ... Method2(...) ;

    // Other methods
    [...]
};
```

ImplObject.cpp

```
[...]
#include "ImplObject.h"

CATImplementClass ( ImplObject,
Implementation,
CATBaseUnknown,
CATNull )

[...]
// TSTIMyInterface adhesion
[...]
... ImplObject ::Method1() { ... }
... ImplObject ::Method2() { ... }

// Other methods
[...]
```

CAA Macros for an implementation class

ImplObject.h

```
... Method1(...);
... Method1(...);
....
```

TSTIMyInterface.h

```
virtual ... Method1(...) = 0;
virtual ... Method2(...) = 0;
```

[Student Notes:](#)

Implementation at work (2/2)

ImplObject1.h

```
[...]
#include "ImplObject.h"

class ImplObject1: public ImplObject
{
CATDeclareClass; ←

public:
    ImplObject1();
    ~ImplObject1();

// TSTIMyInterface1 adhesion
    ... Method3(...);
    ... Method4(...);

// Other methods
[...]
};
```

ImplObject1.cpp

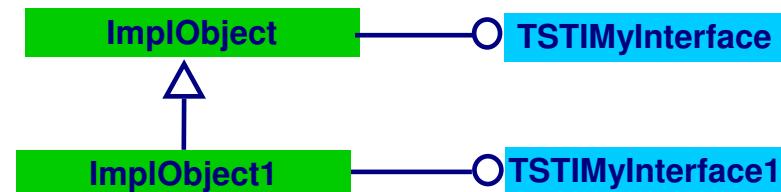
```
[...]
#include "ImplObject1.h"

CATImplementClass ( ImplObject1,
Implementation,
ImplObject,
CATNull )

[...]
// TSTIMyInterface adhesion
[...]
... ImplObject1 ::Method3() { ... }
... ImplObject1 ::Method4() { ... }

// Other methods
[...]
```

CAA Macros for an implementation class with inheritance



The CATImplementClass Macro

Student Notes:

```
CATImplementClass ( <this ClassName>,
< Implementation >,
<its C++ inheritance | CATBaseUnknown>,
CATNull )
```

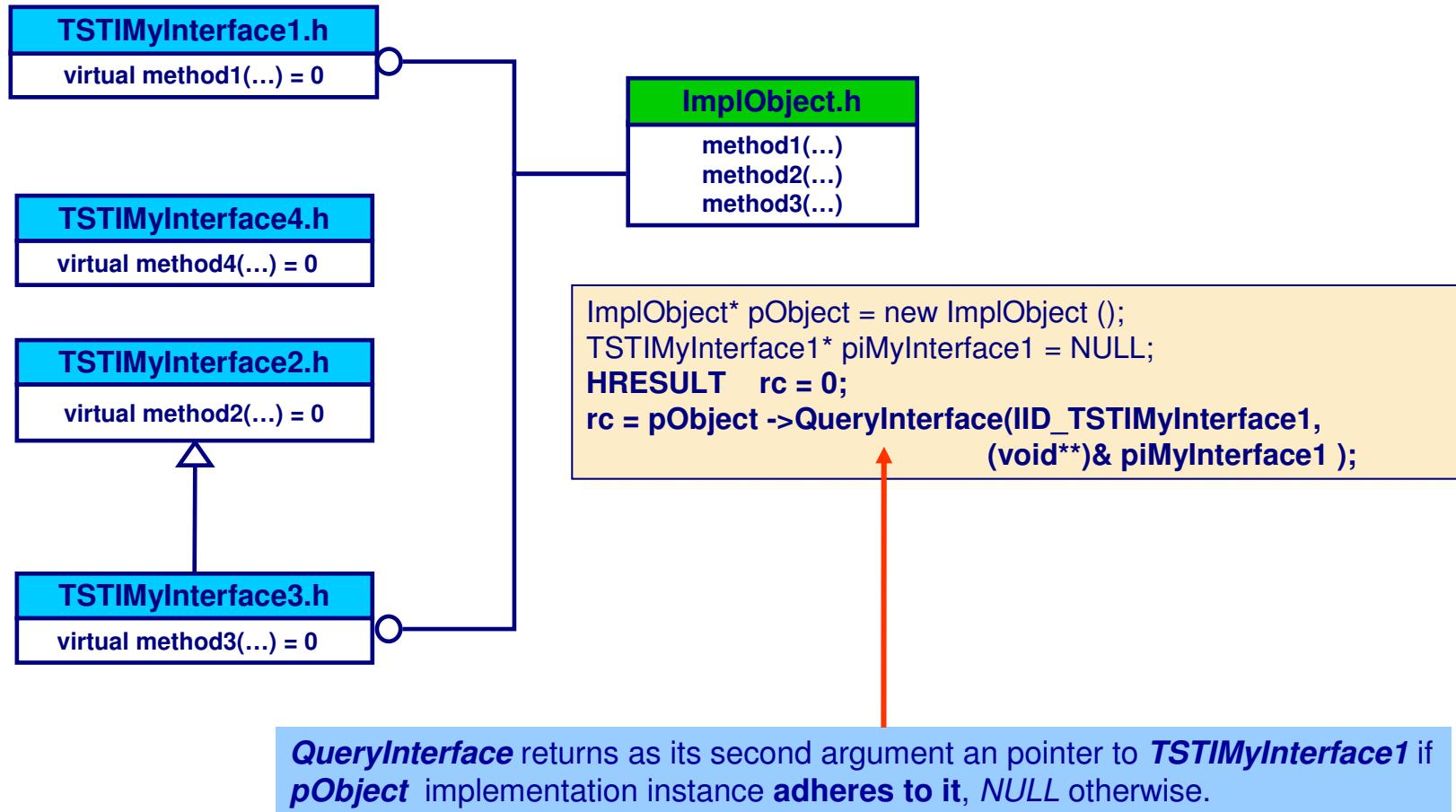
The Name of the implementation

The implementation
It inherits from

Student Notes:

Using Interfaces (1/4)

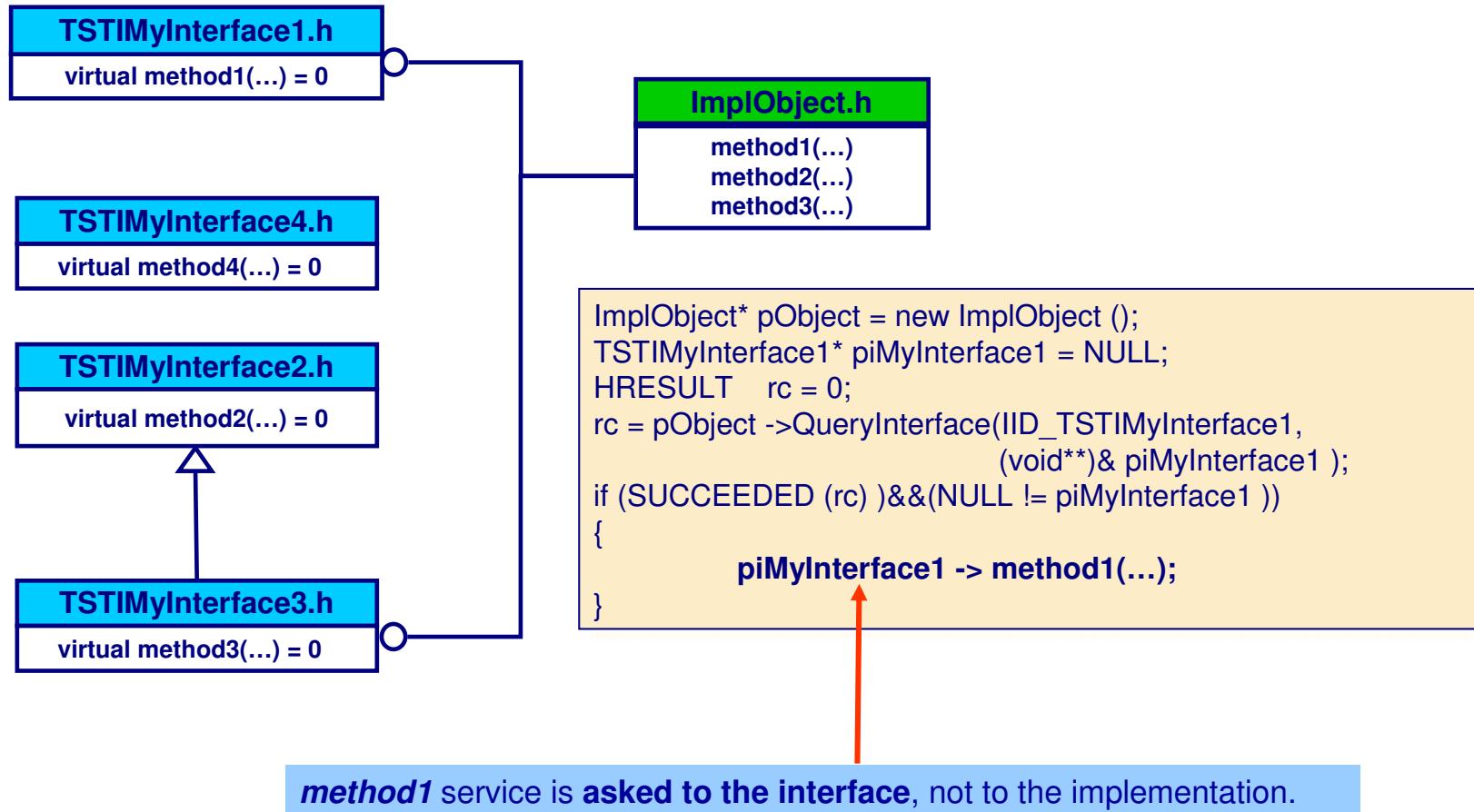
- Getting an Interface from an Implementation



Student Notes:

Using Interfaces (2/4)

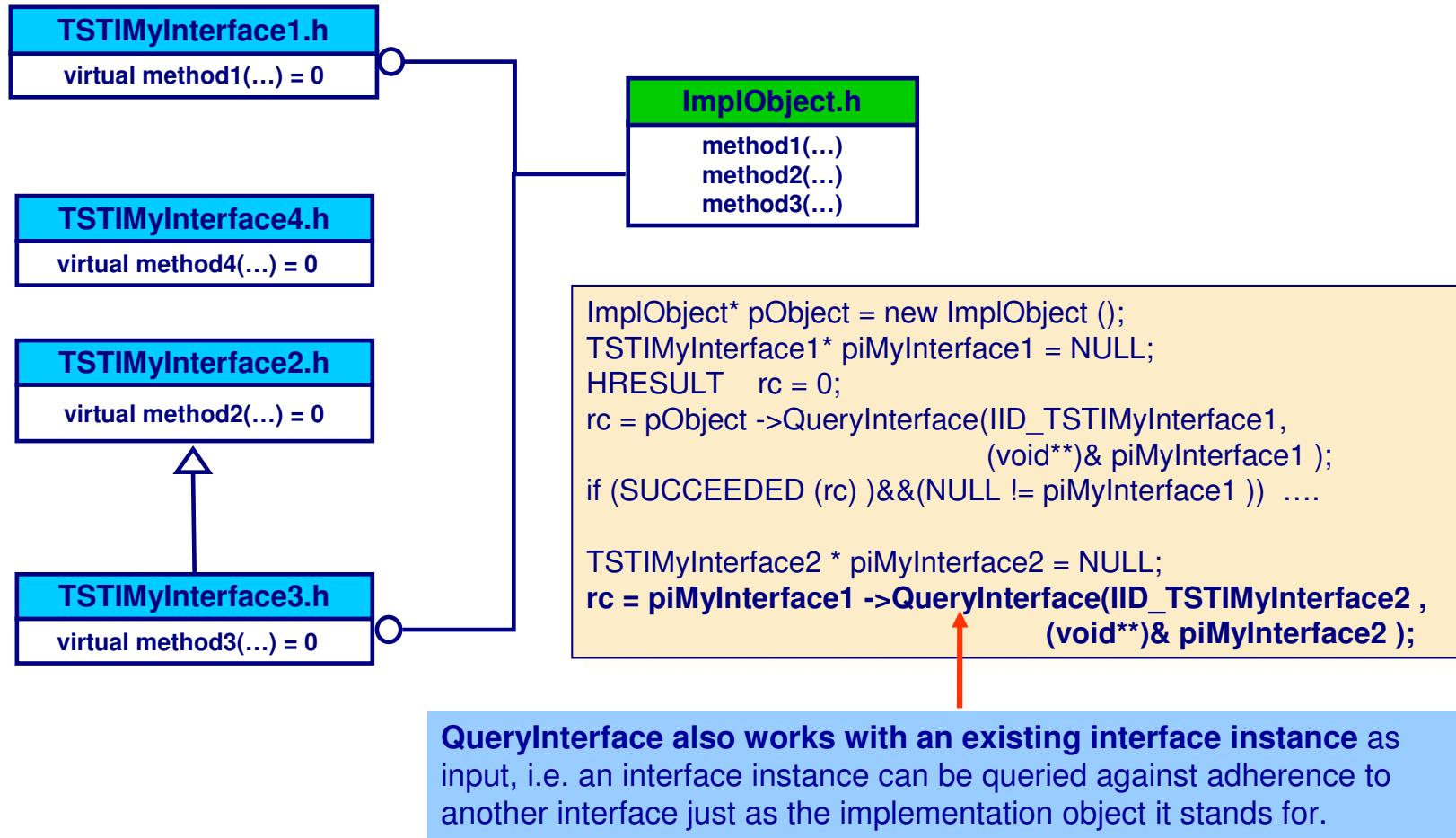
- Using an interface method



Student Notes:

Using Interfaces (3/4)

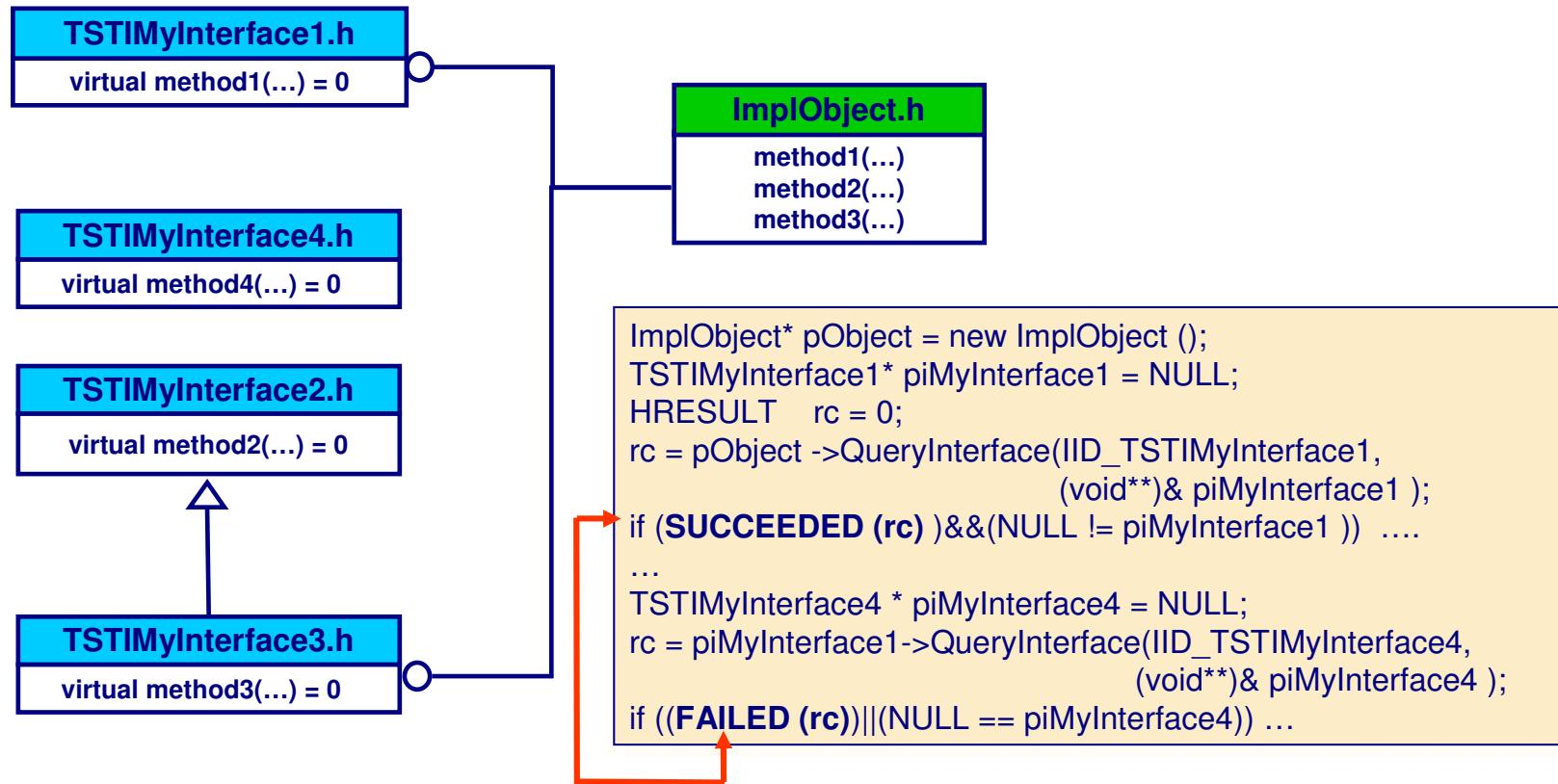
- Getting an Interface from another Interface



Student Notes:

Using Interfaces (4/4)

- Manage error



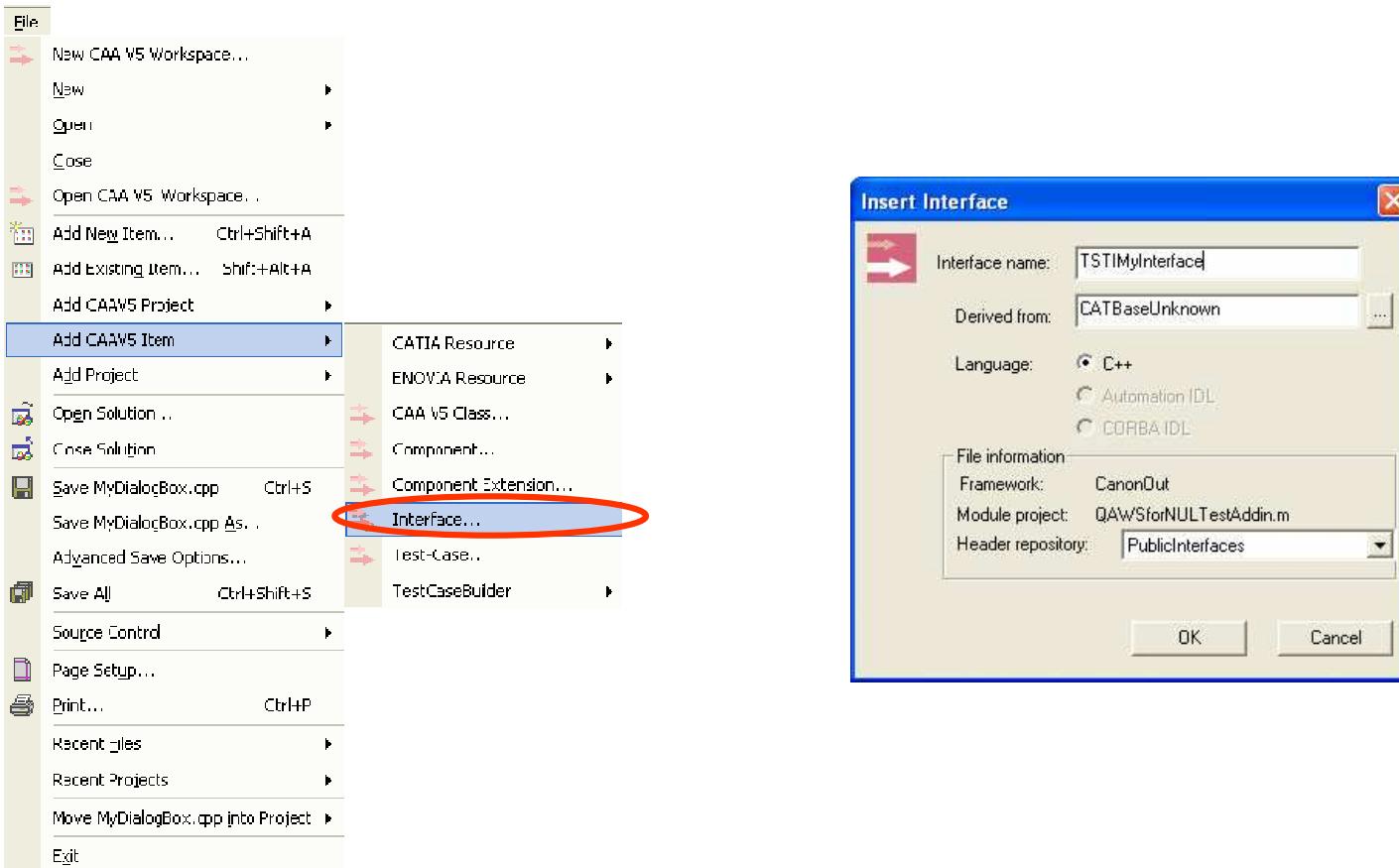
Check that the object adheres to the interface by testing the **HRESULT** return value. If the object doesn't adhere to the interface **HRESULT = E_FAIL** (see **HRESULT** documentation).

[Student Notes:](#)

Create an Interface with CAA V5 wizards in Visual C++



Create TSTIMyInterface interface.

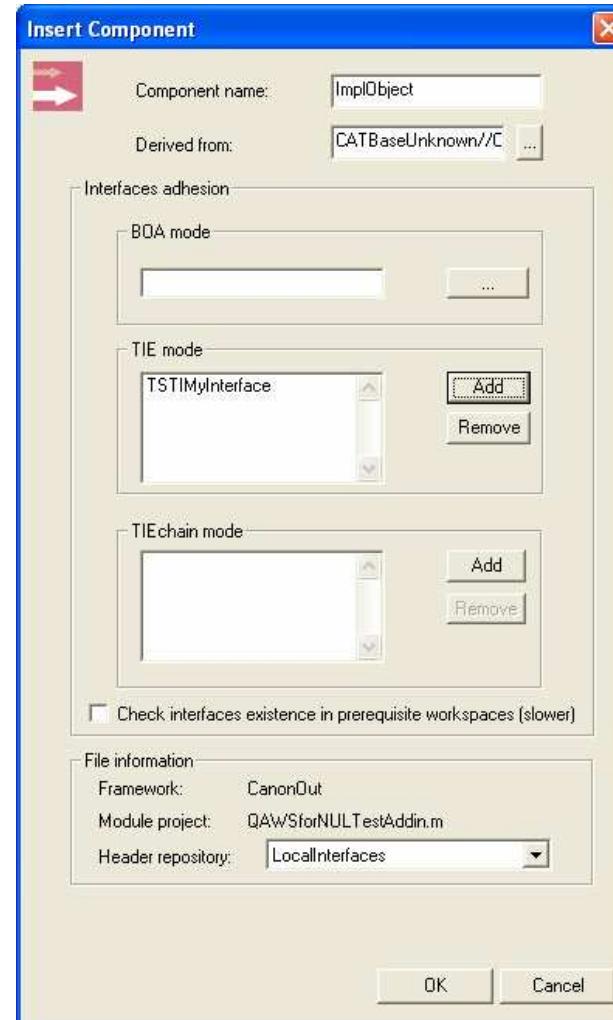
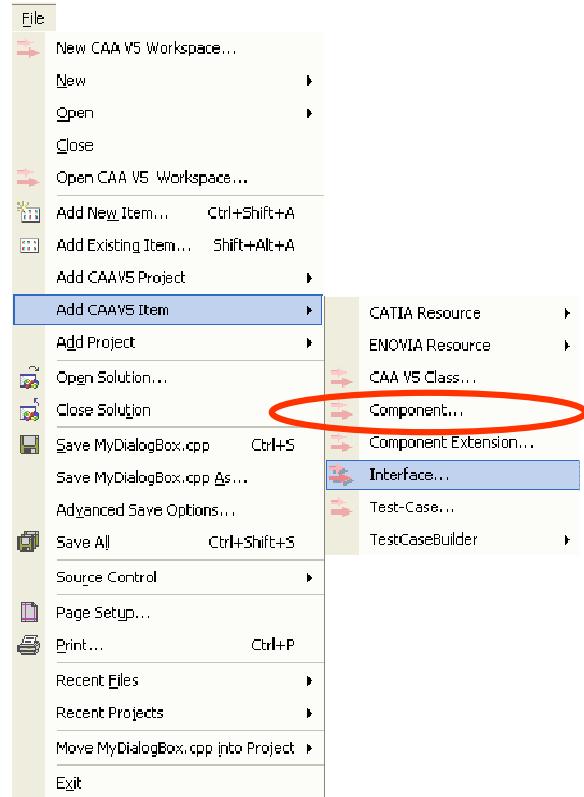


Student Notes:

Create an Implementation with CAA V5 wizards in Visual C++

TSTIMyInterface.h O — **ImplObject.h**

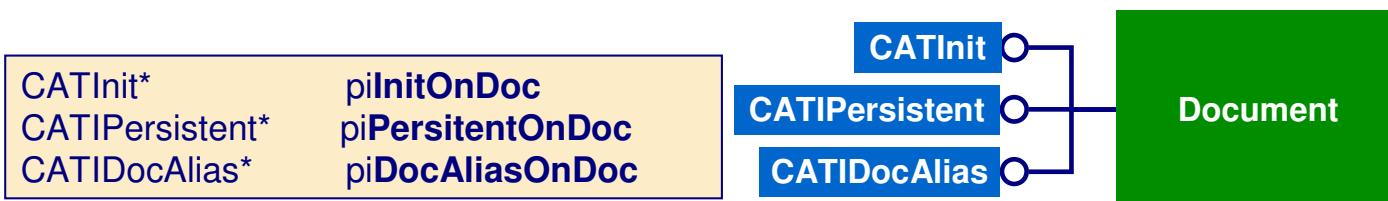
- >Create an implementation that adheres to the TSTIMyInterface interface.



[Student Notes:](#)

Coding Rules (1/3)

- Adopt a consistent naming scheme
- Manipulating the same object through many different interface pointers quickly gets confusing



- Using the same interface to manipulate different objects quickly gets confusing



Student Notes:

Coding Rules (2/3)

- Always test the return value of the `QueryInterface` method before using the returned pointer

```
TSTIMyInterface2 * piMyInterface2 = NULL;
rc = piMyInterface1 ->QueryInterface(IID_TSTIMyInterface2 ,
                                         (void**)& piMyInterface2 );
if (SUCCEEDED (rc) &&(NULL != piMyInterface2) {
    ....
}
OR
if (FAILED (rc) ||(NULL == piMyInterface2) {
    ....
}
```

- Use variable naming convention

◆ Argument prefix: i for input, o for output

◆ Variable prefix:

- p pointer
- pp pointer on pointer
- sp smart pointer
- pi pointer on interface
- a array

<code>TSTIInterface *</code> <code>TSTIInterface *</code> <code>TSTIInterface **</code>	<code>pilInterfaceObject =NULL;</code> <code>aArrayOfInterface2[9] =NULL;</code> <code>ppiInterface3OnObject =NULL;</code>
---	--

[Student Notes:](#)

Coding Rules (3/3)

- Apply following rules for the signature methods

```
virtual HRESULT method ( TSTIMyInterface1 *      ipiMyInterface1,
                         TSTIMyInterface2 **    oppiMyInterface2,
                         TSTIMyInterface3 **    ioppiMyInterface3 ) = 0;
```

OR

```
virtual HRESULT method ( TSTIMyInterface1 *      ipiMyInterface1,
                         TSTIMyInterface2 *&  oppiMyInterface2,
                         TSTIMyInterface3 *&  ioppiMyInterface3 ) = 0;
```

[Student Notes:](#)

Interface Limitation

- **Interface / Implementation pattern :**
 - ◆ Application uses interfaces instead of implementation objects.
 - ◆ The application remains isolated from the implementation.



- Clients should not deal with implementations to avoid coupling.
 - ◆ So we need to encapsulate the implementation instantiation

```
ImplObject* pObject = new ImplObject();  
TSTIMyInterface1* piMyInterface1 = NULL;  
HRESULT rc = 0;  
rc = pObject ->QueryInterface( IID_TSTIMyInterface1,  
                               (void**)& piMyInterface1 );
```

This capability is offered through a special object ...

[Student Notes:](#)

Factory

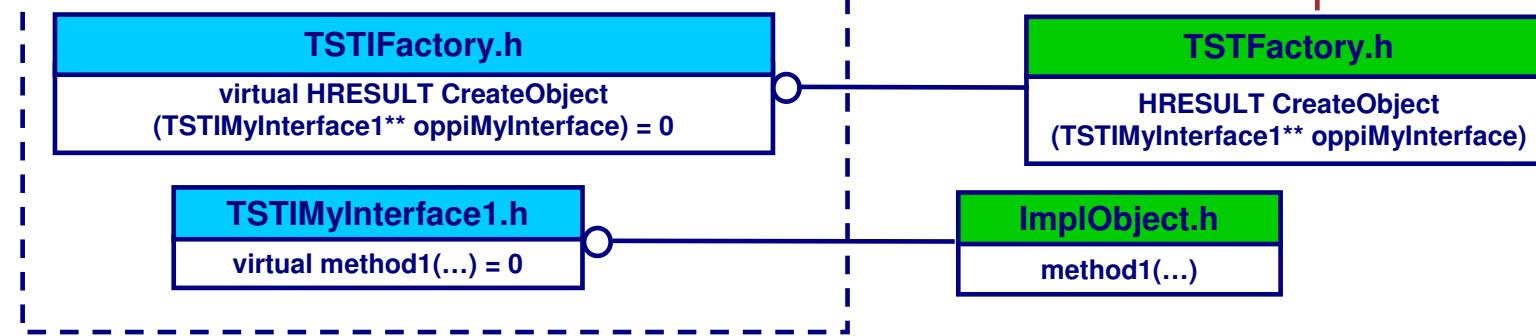
- A factory is a special object that contains methods dedicated to object creation.
- The factory creates an implementation object and returns an interface object on it.
- The factory minimizes the coupling since the client application doesn't manipulate any implementation instance.
- The object creation is centralized which enables a better object management.
- The factory object is instantiated when the application (document, container...) is initialized and retrieved through methods on it.

Factory at work

The **CreateObject** method creates an implementation object and returns an interface on it to the client application.

Client view

```
HRESULT CreateObject (TSTIMyInterface1 ** oppiMyInterface)
{
    (*oppiMyInterface) = NULL;
    ImplObject* pObject = new ImplObject ();
    HRESULT rc = 0;
    rc = pObject ->QueryInterface(IID_ TSTIMyInterface1,
                                    (void**) oppiMyInterface);
    pObject->Release(); pObject = NULL;
    return rc;
}
```



~~ImplObject* pObject = new ImplObject ();
TSTIMyInterface1* piMyInterface1 = NULL;
HRESULT rc = 0;
rc = pObject ->QueryInterface(IID_ TSTIMyInterface1,
(void**)& piMyInterface1);~~

CATIFactory * piFactory = ... ;
TSTIMyInterface1* piMyInterface1 = NULL;
HRESULT rc = S_OK;
rc = piFactory ->CreateObject(&piMyInterface1);

Student Notes:

Exercise Presentation

→ Object Modeler Exercise : Using behavior through interface

And now practice on 2 exercises, to learn about:

- ◆ **Using behavior through interface**

- Set up the environment
- Instantiate objects
- Manipulate objects through interfaces

- ◆ **Define different behaviors with the same interface**

- Implement interfaces
- Query Interface on an object

Student Notes:

CAA V5 Object Modeler: Extension Mechanism

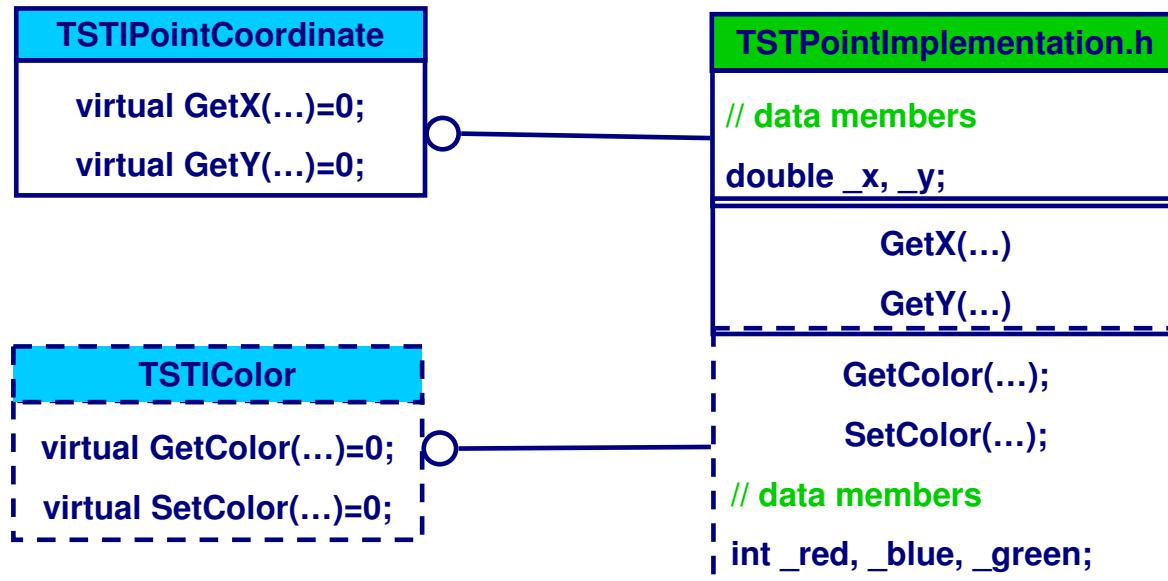
You will learn how to provide existing objects with new behaviors

- Extension
- Extension at work
- Extension Types
- Dictionary
- Extension with CAA V5 wizards

[Student Notes:](#)

Interface / Implementation limits

- How to add new capabilities (behaviors) on an existing object without modifying the implementation object ?



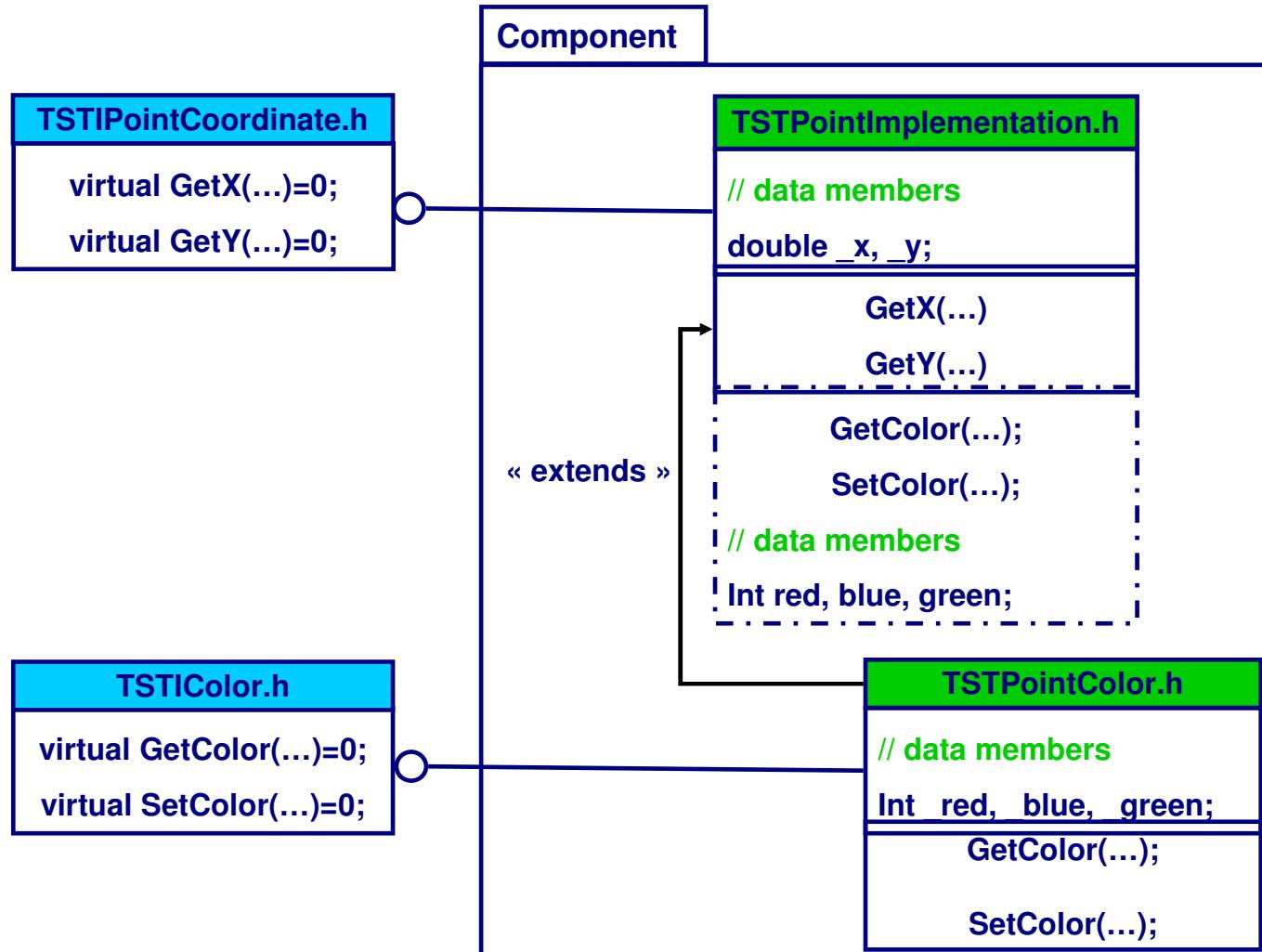
Student Notes:

Extension Definition

- An extension is an object that adds new capabilities to an existing implementation object.
- An extension implements interfaces to create a component
- Component = Base Object + Extension
- An extension can rely on capabilities exposed by its base implementation or other extensions

[Student Notes:](#)

Extension Example



[Student Notes:](#)

Extension at work

ExtObject.h

```
[...]
#include "CATBaseUnknown.h"
class ExtObject: public CATBaseUnknown
{
    CATDeclareClass;           ←----->
public:
    ExtObject() ;
    ~ExtObject() ;
// TSTIMyInterface2 adhesion
    ... Method3(...) ;
    ... Method4(...) ;

// Other methods
[...]
} ;
```

CAA Macros for an extension class

ExtObject.cpp

```
[...]
#include "ExtObject.h"

CATImplementClass ( ExtObject,
DataExtension,
CATBaseUnknown,
ImplObject )
```

**3rd argument useless
for an extension**

```
// TSTIMyInterface adhesion
[...]
... ExtObject ::Method3() { ... }
... ExtObject ::Method4() { ... }

// Other methods
[...]
```

TSTIMyInterface1.h

```
virtual ... Method1(...) = 0;
virtual ... Method2(...) = 0;
```

TSTIMyInterface2.h

```
virtual ... Method3(...) = 0;
virtual ... Method4(...) = 0;
```

ImplObject.h

```
... Method1(...);
... Method2(...);
```

ExtObject.h

```
... Method3(...);
... Method4(...);
```

Student Notes:

Extension Types

CATImplementClass (<this ClassName>,
<its OM type>,
<its OM inheritance | CATBaseUnknown>,
<what it extends | CATNull |>)



- 2 extension types:**
- **DataExtension**
 - **CodeExtension**

Data Extension

- ◆ Can contain methods and data members
- ◆ One extension object for each extended object instance
- ◆ Extension deleted with the implementation object

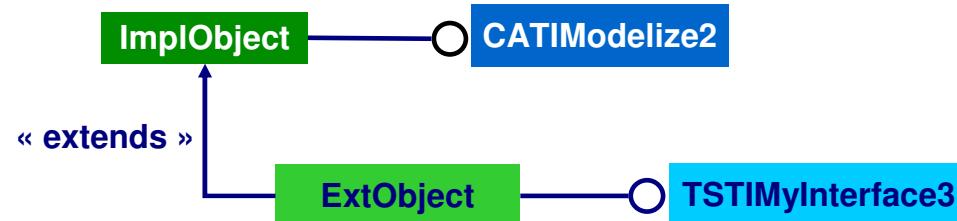
Code Extension

- ◆ Can only contain methods
- ◆ Only one extension object for all extended object instances but for each implemented interface
- ◆ Extension deleted at the end of session

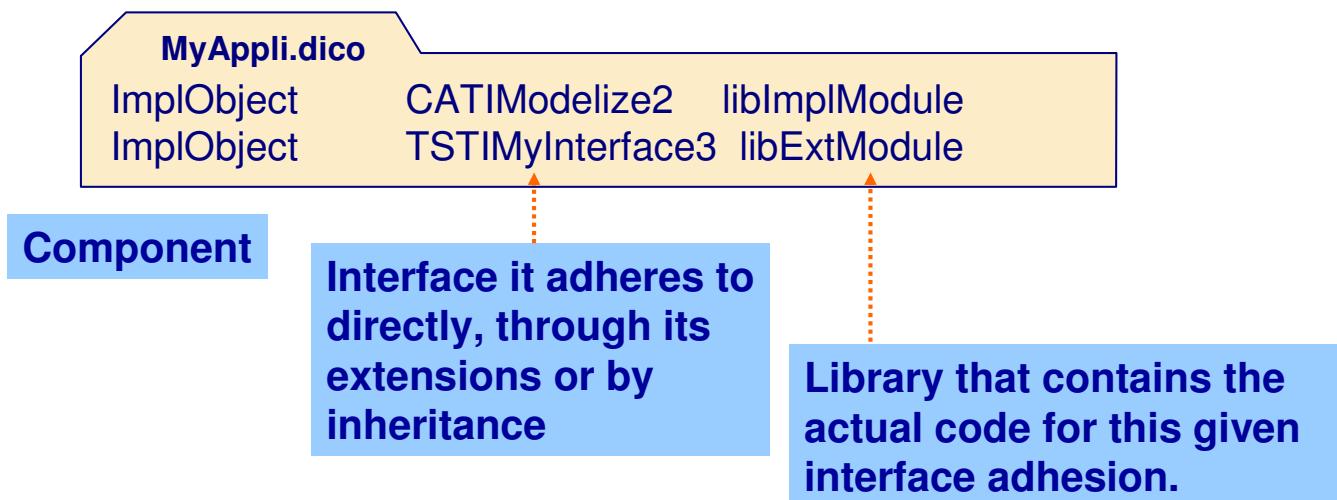
Student Notes:

Dictionary

- A dictionary is required to locate all other interfaces bound to a given implementation or its extensions.



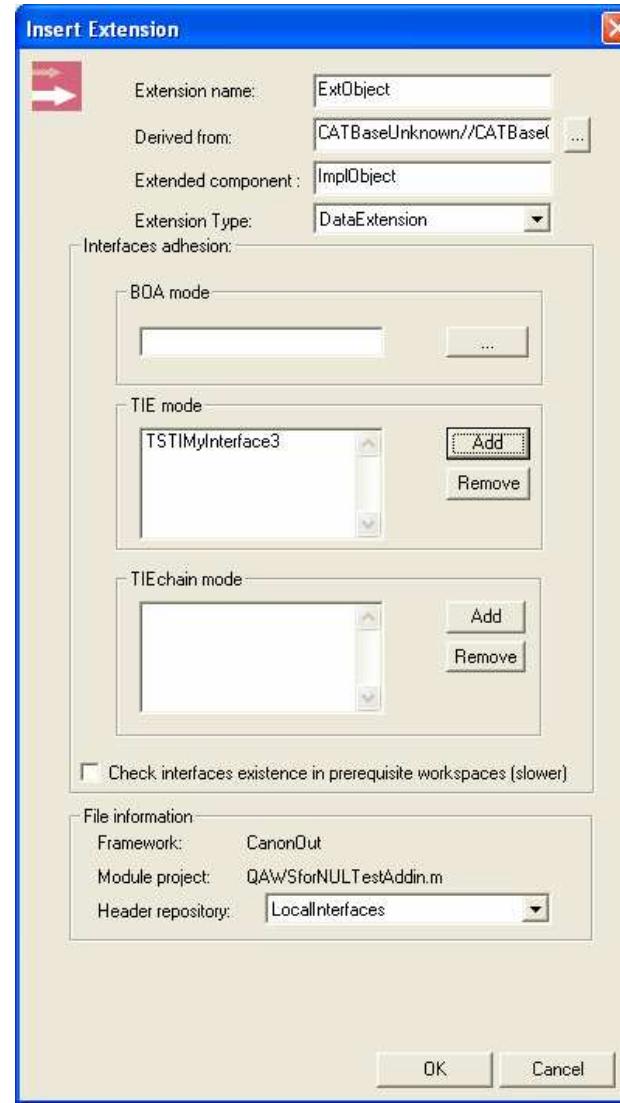
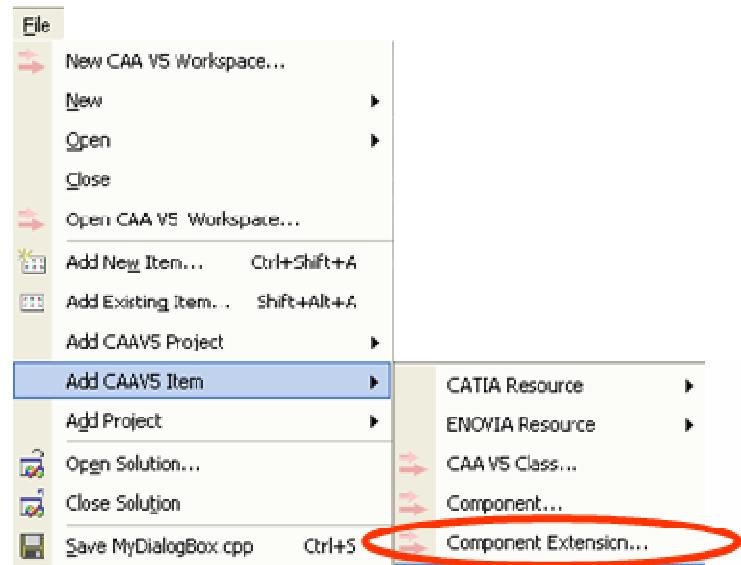
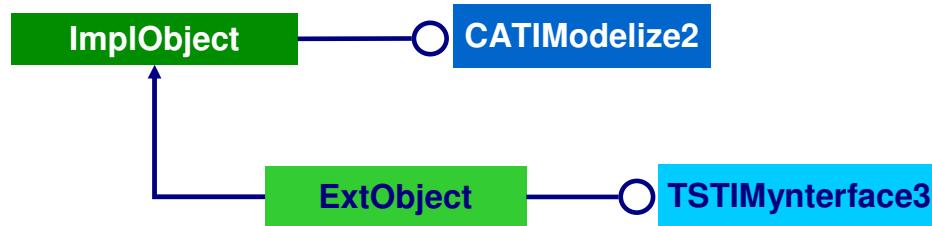
Usually one dictionary per framework stored under
...\\CNext\\code\\dictionary



Student Notes:

Create an Extension with CAA V5 wizards in Visual C++

- Create an extension of `ImplObject` that implements the `TSTIMyInterface3` interface.



[Student Notes:](#)

CAA V5 ObjectModeler: Link between Interface & Implementation

In this lesson you will learn how the link is done between interface objects and implementation objects.

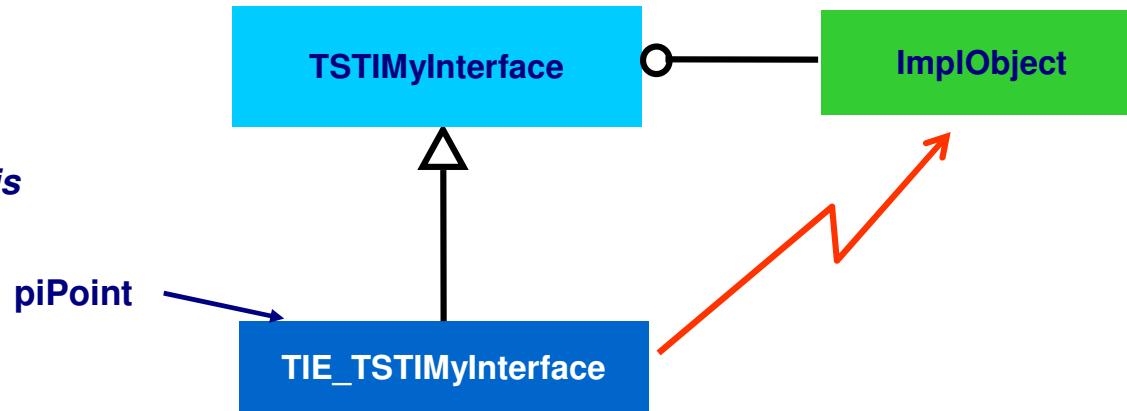
- TIE Definition
- Standard TIE and Chained TIE
- TIE Generation
- BOA
- Standard TIE, Chained TIE and BOA at work

[Student Notes:](#)

TIE Definition

- A TIE is the object that links the interface and the implementation.
- Every request to an interface method is redirected to the corresponding implementation method through the TIE.

A pointer on an interface is in fact a pointer on a TIE object.



Student Notes:

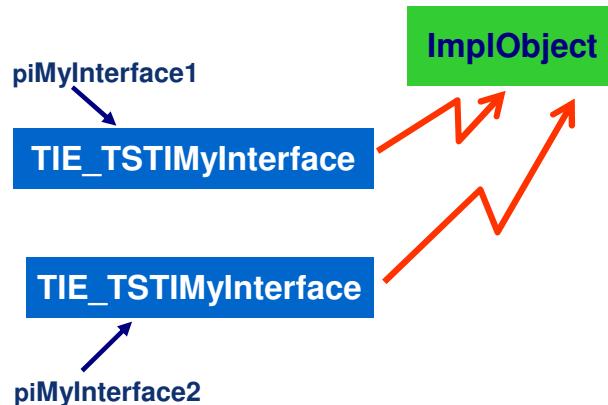
Standard TIE and Chained TIE

Standard TIE

- ◆ A new TIE object is created for each QI even if the TIE already exists
- ◆ No additional data stored in the implementation object

Chained TIE

- ◆ During the QI, try to find an existing TIE
- ◆ Only one TIE allocated but referenced in the implementation (chained list)
- ◆ 1st QI is slower
- ◆ Next QIs are far faster



Student Notes:

TIE Generation

- If the interface is defined in C++, the TIE header file will be generated if you create a file TIE_TSTIxXX.tsRC that just includes the interface header file TSTIxXX.h.
- TIE_TSTIxXX.tsRC is generated by the CAA V5 wizard under Visual C++

<FWInterfaces>/FWItfCPP.m/CATIPoint.tsRC

```
//Code Generated by the CAA Wizard  
//This source file insures the regeneration of the tie TIE_TSTIMyInterface.h  
#include "TSTIMyInterface.h"
```

mkmk

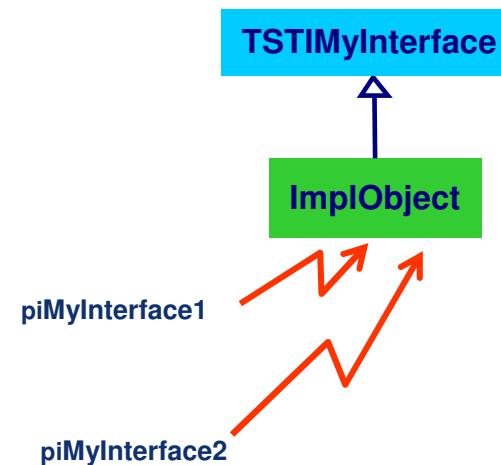
<FWInterfaces>/ProtectedGenerated/intel_a/TIE_TSTIMyInterface.h

...

[Student Notes:](#)

BOA (Basic Object Adapter) Definition

- ❖ Don't create TIE object
 - ◆ Save memory
 - ◆ Direct access to an extension or an implementation
 - ◆ Better CPU performances
 - Polymorphism instead of re-route
- ❖ Implementation object must inherit from the interface
- ❖ Restriction:
 - ◆ An object can implement only one interface (multiple inheritance is forbidden in CAA).
 - ◆ It is not available with code extension type classes.
 - ◆ Some interface adapters don't support the BOA.



Student Notes:

Standard TIE, Chained TIE and BOA at work

ImplObject.cpp

```
[...]
#include "ImplObject.h"

CATImplementClass ( ImplObject,
                    Implementation,
                    CATBaseUnknown,
                    CATNull )

[...]
// Link the implementation to its interface
// -----
//BOA definition
CATImplementBOA(TSTI1,ImplObject);

//TIE or TIEchain definitions
#include "TIE_TSTI2.h"
TIE_TSTI2(Comp2);

#include "TIE_TSTI3.h"
TIEchain_TSTI3(Comp2);

... ImplObject ::Method1() { ... }
... ImplObject ::Method2() { ... }

// Other methods
[...]
```

CATImplementBOA(...)
macro is used for a BOA

TIE_xxx macro is used
for a standard TIE

TIEchain_xxx macro is
used for a standard TIE

[Student Notes:](#)

CAA V5 Object Modeler: Component

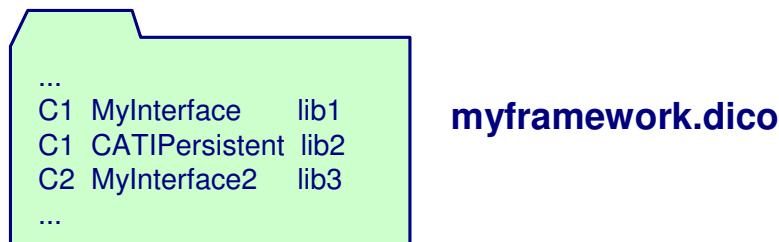
In this lesson you will learn how to instantiate types unknown at build time

- Component
- Late Type
- Late Type and Inheritance
- Extension
- CAA V5 Object Browser

Student Notes:

Component

- A Component is defined by a base object which can be:
 - An Implementation
 - A Late Type
- It can adhere to one or several interfaces thanks to the extension mechanism.
- These extensions are defined in a dictionary (`myframework \ CNext \ code \ dictionary \ myframework.dico`)



Late Type

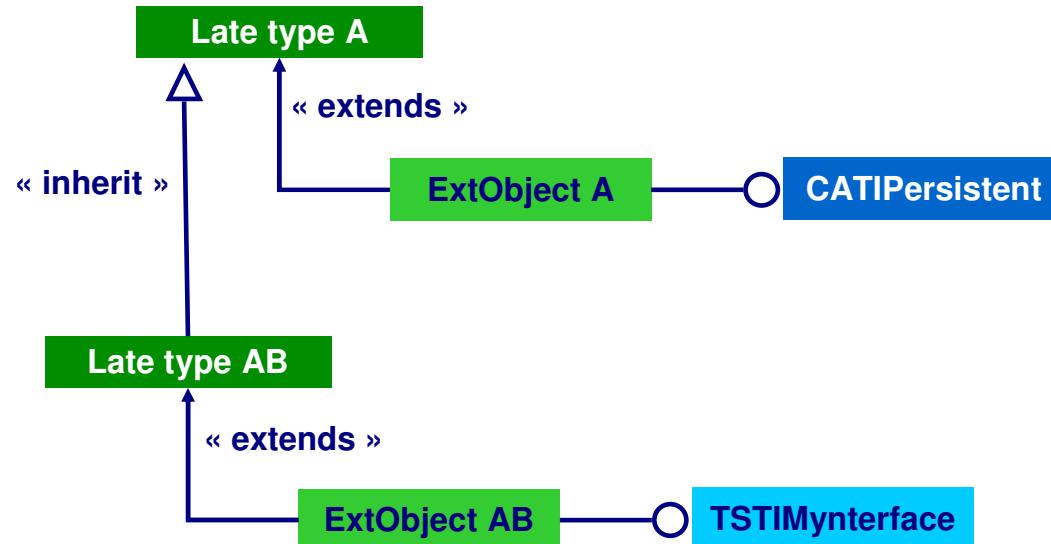
Student Notes:

- A Late Type is a character string.
- It is a mechanism which enables the instantiation of components by name at run time.
- Different kind of objects have a late type (we will speak about these objects later) :
 - ◆ Feature
 - ◆ Document (CATPart, CATProduct, ...)
 - ◆ Container
 - ◆

Student Notes:

Late Type and Inheritance

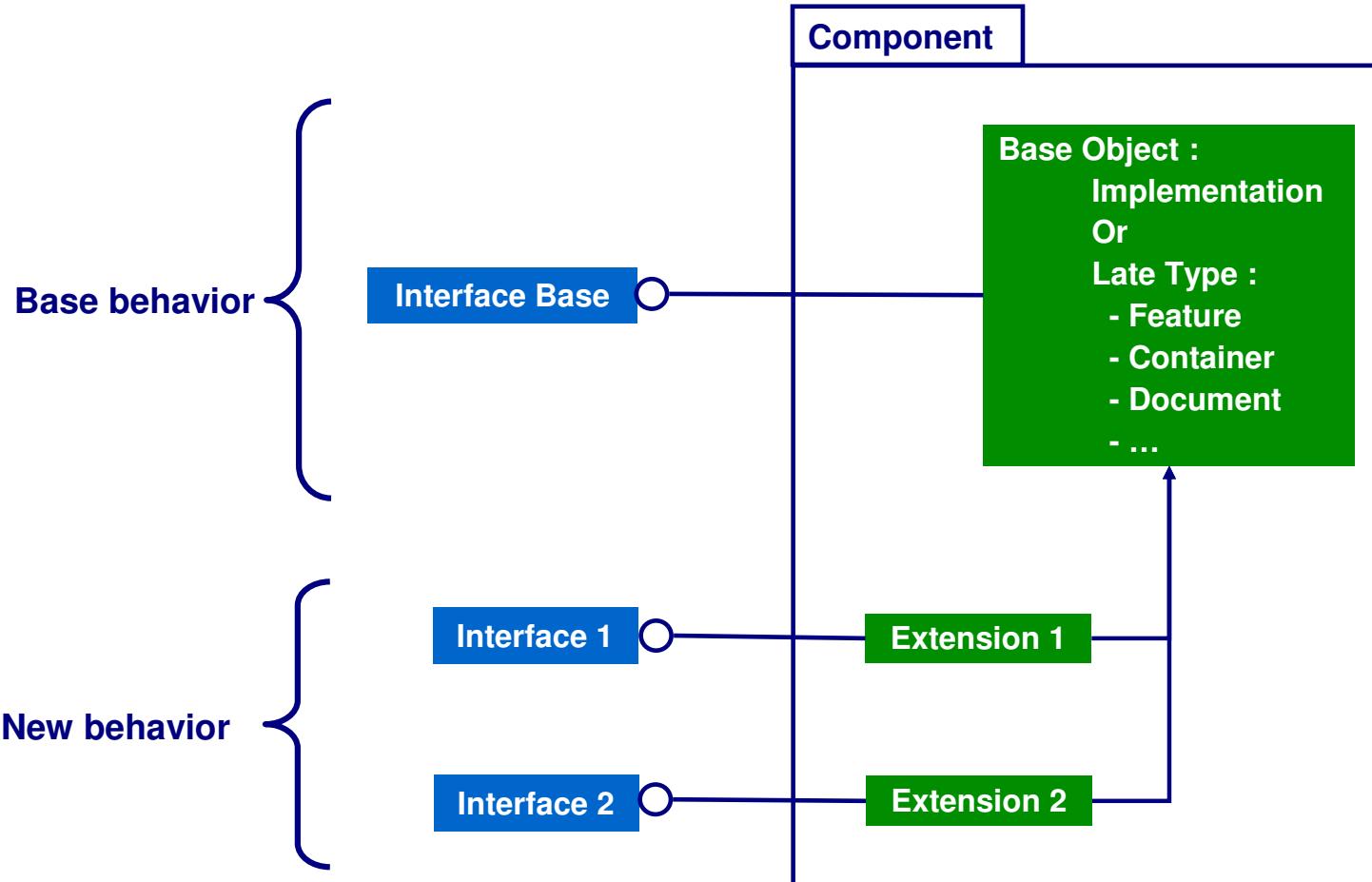
- In the same way than real type, late types support inheritance.



Through inheritance, AB gets A behaviors using QueryInterface

[Student Notes:](#)

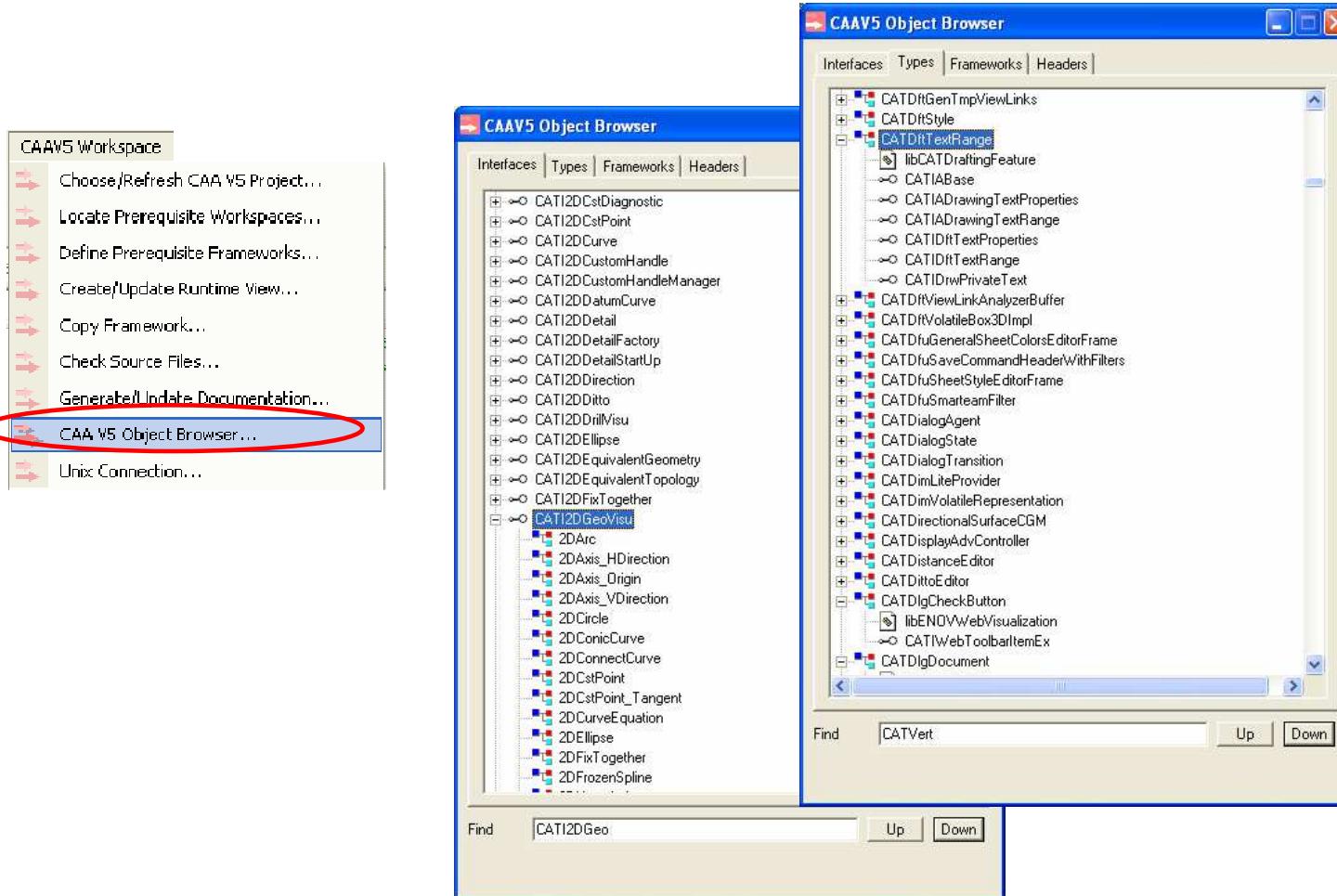
Extension



Student Notes:

CAA V5 Object Browser

- You can see the Late Type adhesion with the CAA V5 Object Browser.



[Student Notes:](#)

CAA V5 ObjectModeler: Object Life Cycle Management

You will learn how to manage the life cycle of interfaces and implementations through reference counting and smart pointer techniques.

- CAA V5 ObjectModeler Rules
- Managing the Object Life Cycle
- Reference Counting
- Smart Pointers
- Do and do not
- mkCheckSource

CAA V5 ObjectModeler Rules

- Implementation should be manipulated by applications only through interfaces.
- An implementation can only be deleted when its last pointing interface is deleted.
- The CAA V5 ObjectModeler deletes automatically implementation objects.

Student Notes:

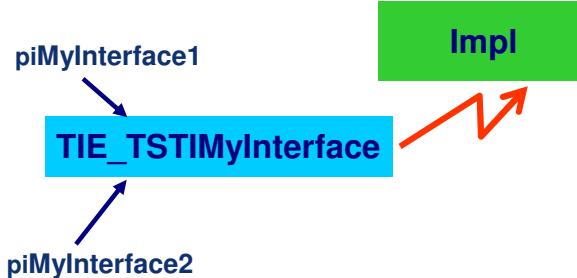
But who should delete objects and how ?

Managing the Interface Life Cycle

- We can have many references on the same interface at the same time.

```
delete piMyInterface1 ;  
// Access to piMyInterface2 ;
```

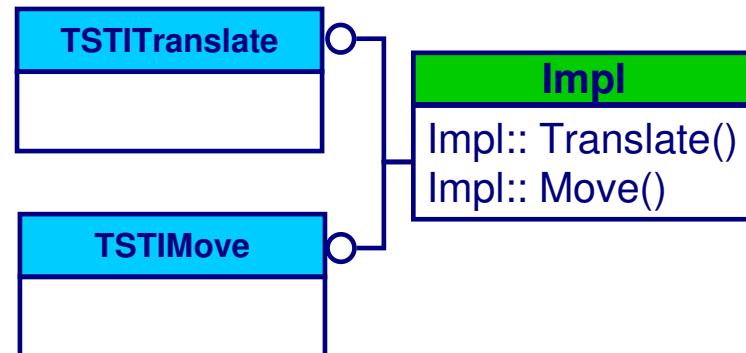
Don't use delete() on Interfaces .



- We usually holds more than one reference to an implementation at a time

```
piTSTIMove->DeleteImpl();  
// Access to piTSTITranslate
```

Don't use `delete()` on Implementations .

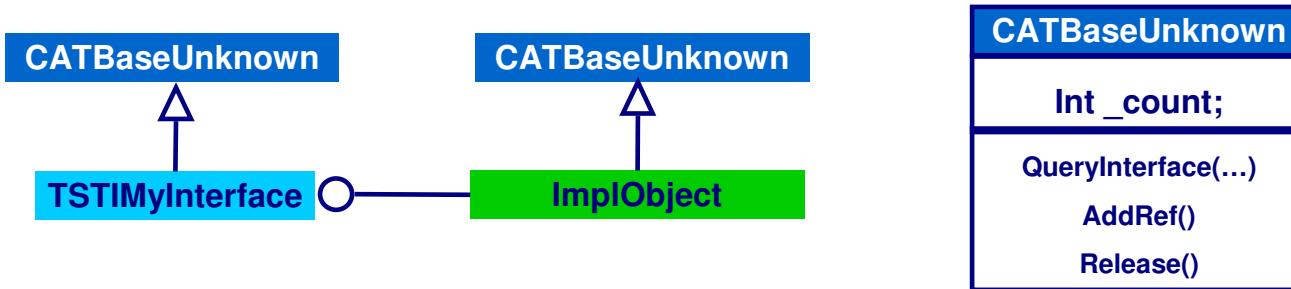


ObjectModeler offers two mechanisms that secure the existence of an interface used by objects.

[Student Notes:](#)

Reference Counting

- Interface offers special methods for locking itself from deletion by others
- A client of an interface increments the reference count on the interface while it needs it.
- A client decrements the reference count when the interface is no longer needed.
- The removal of the last reference causes the deletion of the interface by ObjectModeler.



[Student Notes:](#)

Reference Counting at work

... QueryInterface does an **AddRef()** to secure the interface pointer before returning it to the caller.



```
[...]
CAAIPoint * piPoint = NULL;
hr = piOnPointObject->QueryInterface(IID_CAAIPoint, (void**)& piPoint) ;
piOnPointObject ->Release(); // Done with old interface
piOnPointObject = NULL;

piPoint ->GetX( ... ) ;

piPoint ->Release() ; // Done with last interface
piPoint = NULL;
[...]
```

Remark : a pointer copy does not increase the reference count, we must call AddRef()

Student Notes:

Factory and Reference Counting

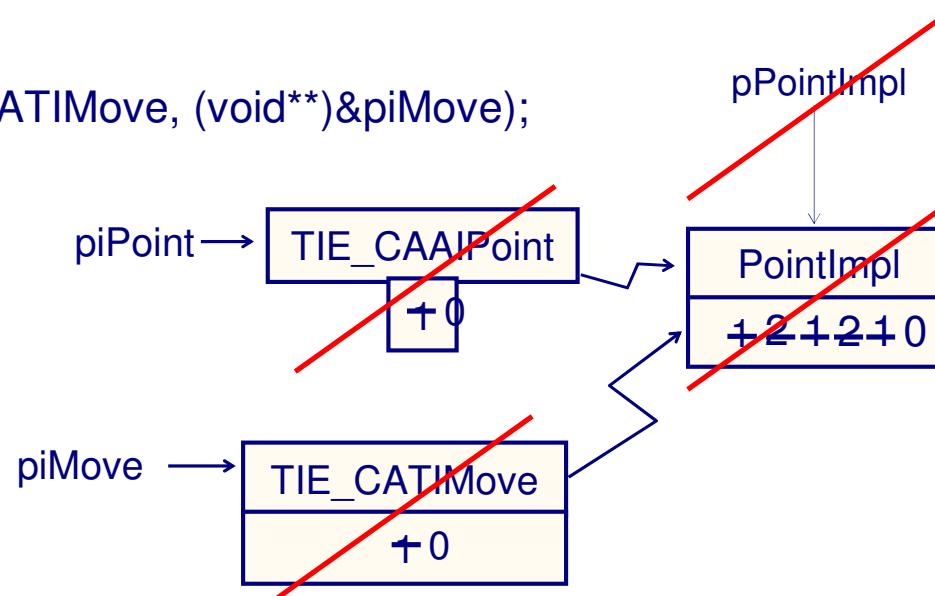
```
CAAIPoint *piPoint = factory->CreatePoint( );
```

```
↳ PointImpl *pPointImpl = new PointImpl( );
    pPointImpl->QueryInterface( IID_CAAIPoint, (void**)&piPoint);
    pPointImpl->Release( ); pPointImpl = NULL
    return piPoint;
```

```
piPoint-> QueryInterface(IID_CATIMove, (void**)&piMove);
```

```
piPoint->Release( );
piPoint = NULL;
```

```
piMove->Release( );
piMove = NULL;
```



Student Notes:

Reference Counting: Golden Rules

- Methods returning a pointer to an interface (ex: `QueryInterface`) should always do an `AddRef()` on it prior to returning
- Clients receiving such a pointer should always `Release()` it after usage
 - ◆ Tip: write the `Release()` code just after the function call that created the interface pointer and « push » it with usage code
 - ◆ Tip: when using `QueryInterface()`, you usually `Release()` the previous interface after getting the new one

Student Notes:

Smart Pointers (also called Handlers)

- A smart pointer is a class associated to an interface, that behaves like an interface pointer, with additional automatic reference counting
- Smart pointers for interface CATIxxx are of type CATIxxx_var and are defined in the same header file.
- They manage reference counting through code in destructor, constructor, assignment operator, etc.

Student Notes:

Smart Pointers Arithmetic

```
CATIVisu_var           spiVisuOnObject ;
CATITransform_var      spiTransOnObject ;
```

Operator	Example	Meaning
->	spiVisuOnObject ->Draw()	Dereference. Access a public member of the interface. It makes the usage of interface very similar to a regular C++ pointer on implementation. NB: do not use the "." operator with handlers.
=	spiVisuOnObject = spiTransOnObject	Assignment. Make aCATIVisu refer to the same object as aCATITransfor. Since those handlers are not of the same type, it does only an AddRef if right value is a subtype of left value. If not, it runs a QueryInterface on right value. Therefore the result can be a NULL handler.
()	spiVisuOnObject(spiTransOnObject)	Copy Construction. Handlers preferred way of performing a QueryInterface(). Result can also be a null_var handler.
==	if spiVisuOnObject == spiTransOnObject	IsEqual. Test if two interface instances are dealing with the same implementation. Thanks to NULL_var that defines the NULL handler value.
!	if (! spiVisuOnObject)	Is Null. Test if the interface pointer is not Null

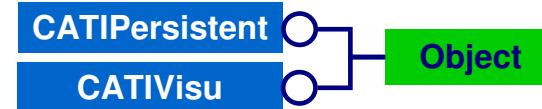
Student Notes:

Smart Pointers or AddRef/Release?

- ❖ When a method returns an interface pointer
 - ◆ Avoid using a handler to get the result
- ❖ When a method returns a handler
 - ◆ Always use a handler to get the result
- ❖ You can use handlers in methods that ask for interface pointers as input parameters (parameters IN)
- ❖ Always use handlers from the correct type in methods that ask for references on handlers (parameters INOUT)

[Student Notes:](#)

Do and Do Not (1/2)



- Considering the method: **CATIVisu_var GetParentVisu();**

Don't do



```
CATIVisu_var spVisuOnX;  
spVisuOnX = GetParentVisu();  
spVisuOnX ->Release();
```

Instead Do



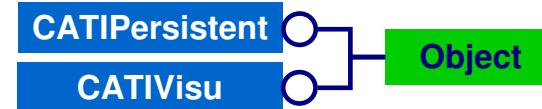
```
CATIVisu_var spVisuOnX;  
spVisuOnX = GetParentVisu();
```



In that case the returned value is a smart pointer that increments the reference count upon creation and decrements it upon deletion.

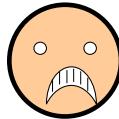
[Student Notes:](#)

Do and Do Not (2/2)



- Considering the method: **CATIVisu* GetVisu();**

Don't do



```

CATIPersistent_var spPersiOnX=NULL_var;
spPersiOnX = GetVisu();
If (NULL_var!= spPersiOnX)
{
  spPersiOnX->Method()
  ...
}
  
```

Instead Do



```

CATIVisu *pVisuOnX = GetVisu();
If (NULL!= pVisuOnX )
{
  CATIPersistent_var spPersiOnX (pVisuOnX );
  pVisuOnX ->Release(); pVisuOnX =NULL;
  ...
}
  
```



Memory Leak because the volatile pointer **CATIVisu* is not released !**

Student Notes:

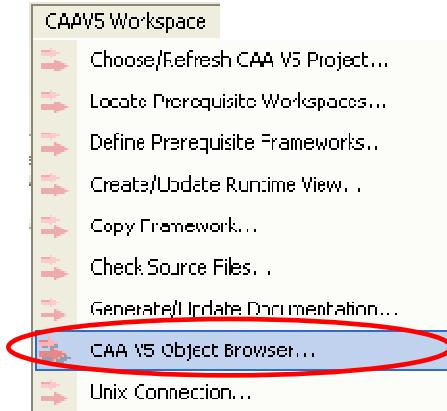
mkCheckSource

- **mkCheckSource (C++ Source Checker)** is a C++ static controller which analyzes a program and tries to discover potential defects or logical errors without executing the program.
- Operating at the source stage in the application development cycle, early checking against C++ coding rules will ensure better stability and reduce defect.
- mkCheckSource provides a full HTML report with hyperlink to defaulter C++ source.

[Student Notes:](#)

mkCheckSource at work

- This command Checks :
 - ◆ AddRef / Release rules
 - ◆ Callbacks rules
 - ◆ Pointer Lifecycle rules
 - ◆ Exceptions rules
 - ◆ C++ rules
 - ◆ Miscellaneous rules
- This command generates a html file with
- The UNIX command is :
 - ◆ Command :
 - mkCheckSource
 - ◆ Option
 - If you want to check only a framework : -f framework_to_analyze
 - To Specify the html file : -html " MyWorkSpace\mkCheckSourceOutputDirectory "



Student Notes:

Exercise Presentation

→ Object Modeler Exercise : Using behavior through interface

And now practice on 2 exercises, to learn about:

- ◆ Extending Point's behavior through Extension mechanism
 - Extend object's behavior

- ◆ Managing the objects life cycle
 - Increment / Decrement the reference counter

[Student Notes:](#)

CAA V5 Specification Modeler

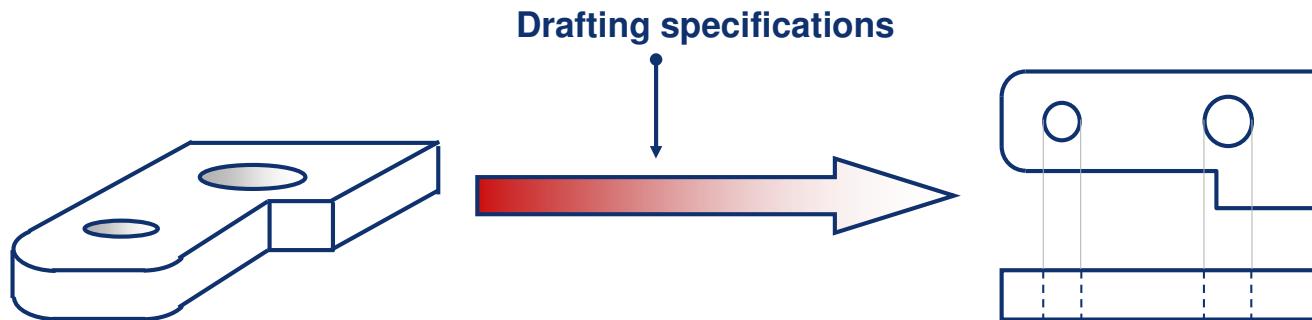
In this course you will be introduced to specification modeling and you will discover the provided modeler for such a modeling

- Reason Of Using Specifications
- Object Specs Modeler Objectives
- Feature Structure
- Document architecture

[Student Notes:](#)

Reason Of Using Specifications

- allows users to accelerate the art-to-part process by capturing specifications and generating results



- Benefits:**
 - Results are generated faster
 - Company rules and know-how are captured
 - Result is explainable
 - Designers do design, not clerical tasks

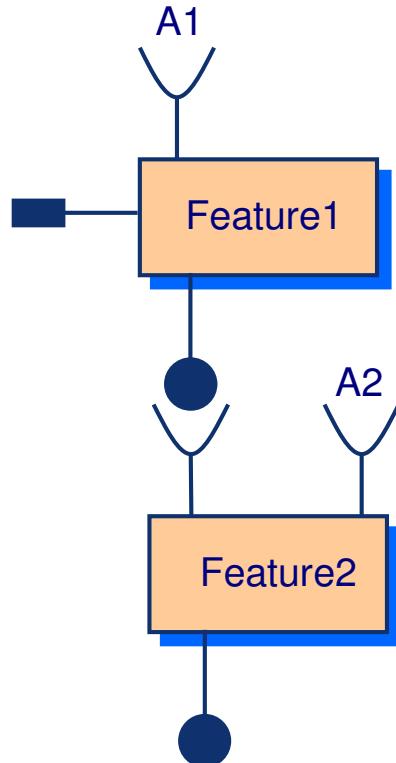
[Student Notes:](#)

ObjectSpecsModeler Objectives

- Provide an infrastructure for Specification/Result management, and therefore associativity
 - ◆ A generic mechanism for change propagation
 - ◆ Persistency
- Provide a standard Specification/Result data structure called **FEATURE**
 - ◆ This data structure is used by every Dassault Systèmes applications
 - ◆ Clients and partners can define their own features
 - ◆ Dassault Systèmes will treat on the same way DS and clients features

[Student Notes:](#)

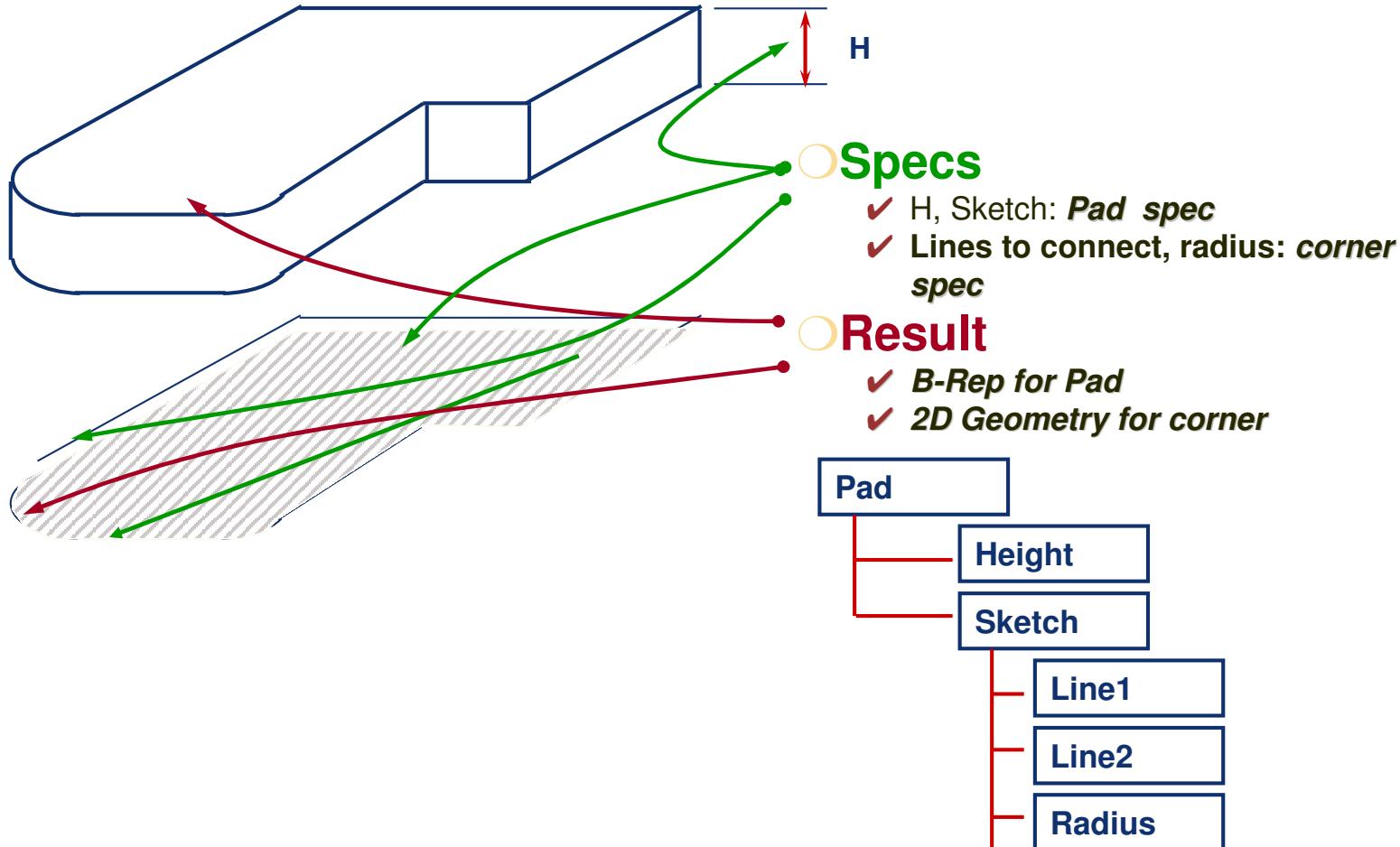
Feature Structure



- A feature is a basic object to store data
- A Feature is defined by a Late Type (its name) and its attributes.
- An attribute stored a data that can be of a simple type (string, double, boolean, octet, integer), a feature type or a list of those types.
- An attribute is defined by its quality (its role in the build process and the update mechanism)
 - ◆ An OUTPUT attribut store the result data
 - ◆ An INPUT attribut store the specification data
 - ◆ An NEUTRAL attribut store any other data
- The update mechanism is only called when an output or an input attribute has been changed

[Student Notes:](#)

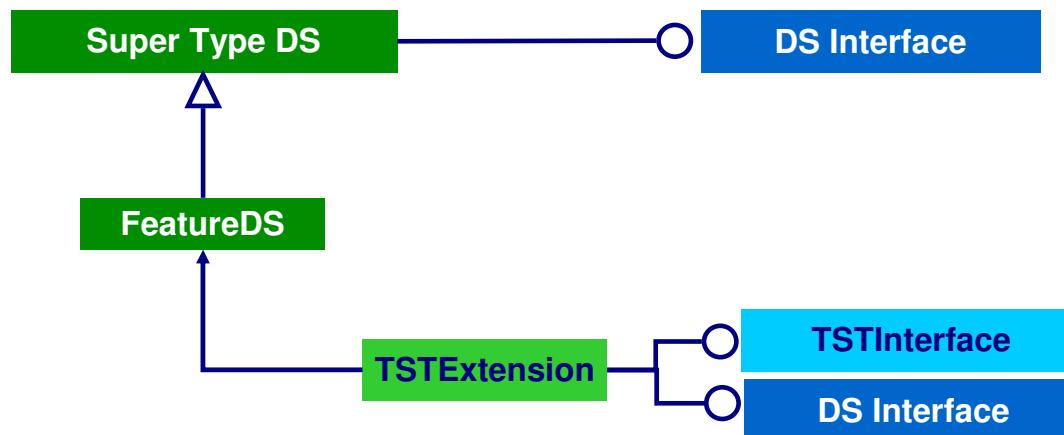
Specification and Result Management Example



[Student Notes:](#)

Feature and Late Typing

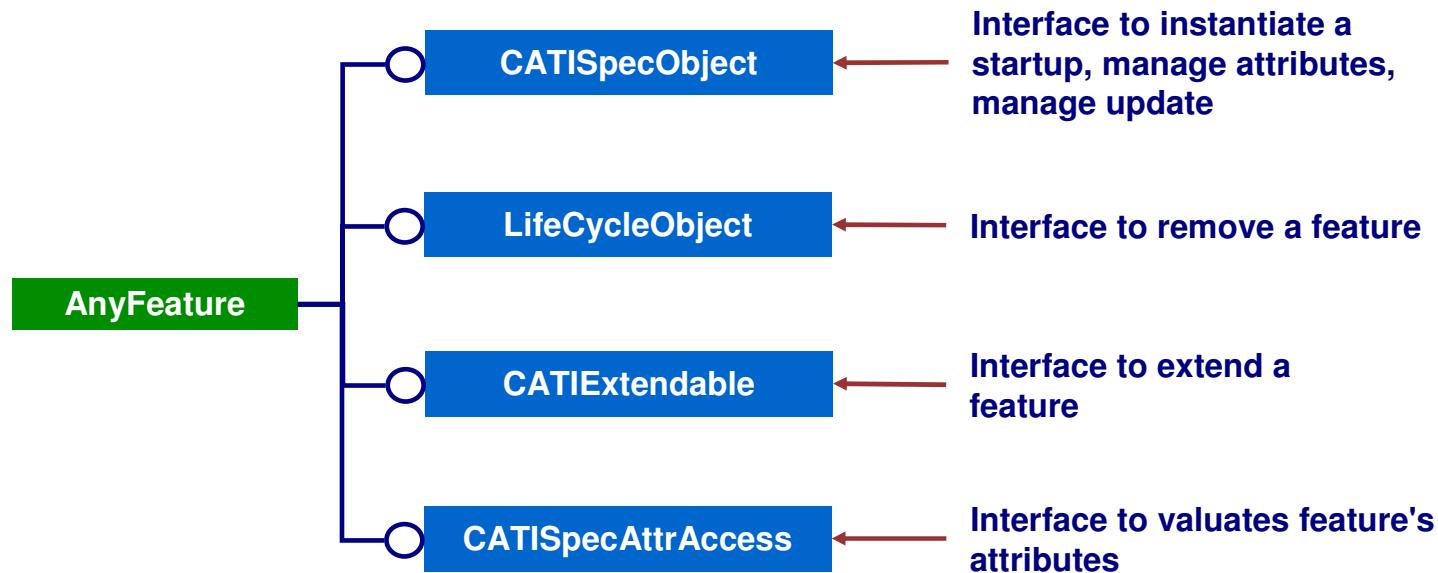
- Because a feature is a Late type
 - ◆ Behaviors are provided through the extension mechanism (an Object Modeler service) following the Interface / Implementation pattern.
 - ◆ Inheritance between features is allowed (see Late type inheritance)
- New feature will inherit both behaviors and attributes of its super type



[Student Notes:](#)

Feature Instances

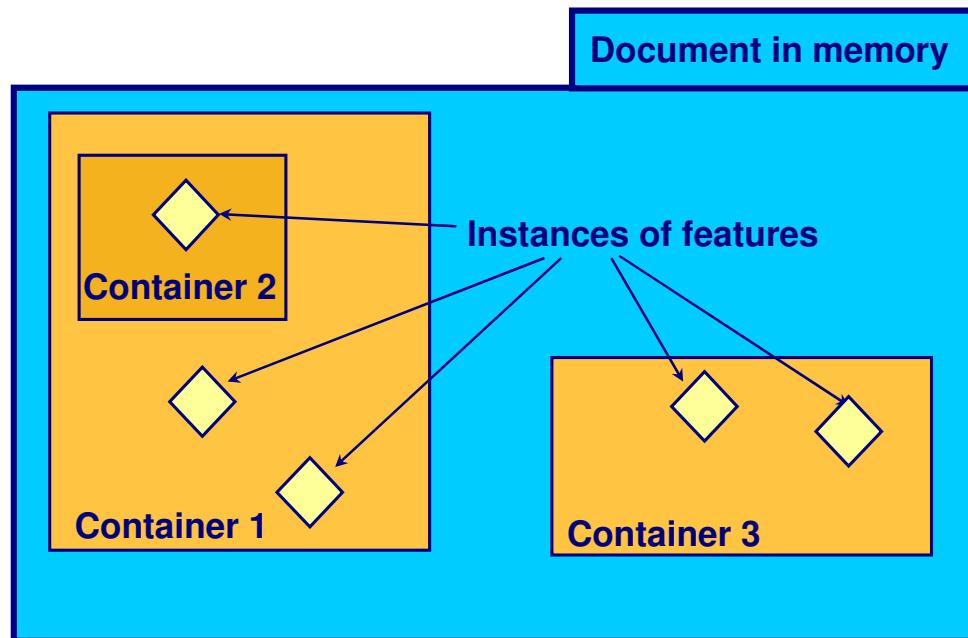
- Every feature is an instance of a **CATSpecObject**
 - It implements several DS interfaces



[Student Notes:](#)

Document architecture and Feature Persistency

- A document is a set of containers.
- To be persistent a feature must be instantiated in a container. This container belongs to a document.
 - ◆ Saving the document will in turn save the feature.



Student Notes:

CAA V5 Mechanical Modeler: Introduction

You will learn what is specific to a mechanical feature, and how the link with the geometry and the topology is managed.

- Objectives of the CAA V5 Mechanical Modeler
- Frameworks
- The CATPart document
- The mechanical features

Student Notes:

CAA V5 Mechanical Modeler Objectives (1/3)

- **Links the Geometric Modeler that manages results with the ObjectSpecsModeler that manages specifications.**

- **Provides the Infrastructure for Mechanical Applications:**
 - ◆ **Applications that create new mechanical specifications that generate geometric results:**
 - Ex: Part Design, Generative Shape Modeling, Sheet Metal, ...

 - ◆ **Applications that just reference the BRep of a part or an assembly and/or read geometric information:**
 - Ex: Assembly Design, Drafting, Generative Part Stress Analysis, IGES interface

Student Notes:

CAA V5 Mechanical Modeler Objectives (2/3)

- Provide a hierarchy of mechanical features and their corresponding interfaces, to be used as is or derived.
 - ◆ Solid or surface or wireframe features
 - ◆ Datum feature, a poor feature that encapsulates topology and geometry without any specification
 - Ex: data coming from neutral format, like IGES or CopyAsResult from V4, ...
 - ◆ Constraint features
- Provide a stable access to the sub-elements of a Mechanical Part.

Student Notes:

CAA V5 Mechanical Modeler Objectives (3/3)

- Provide the CATPart document and its dedicated workshop with some generic interactive commands:
 - ◆ Update with a specific error management
 - ◆ Delete/Cut/Copy/Paste, Reorder, Activate/Deactivate, ...
 - ◆ Manage the intermediate states of the part (Smart concept)
 - ◆ 3D compass use
- Manage the Visualization of the Part
 - ◆ Visualization strategy
 - ◆ Decode

[Student Notes:](#)

Framework Architecture

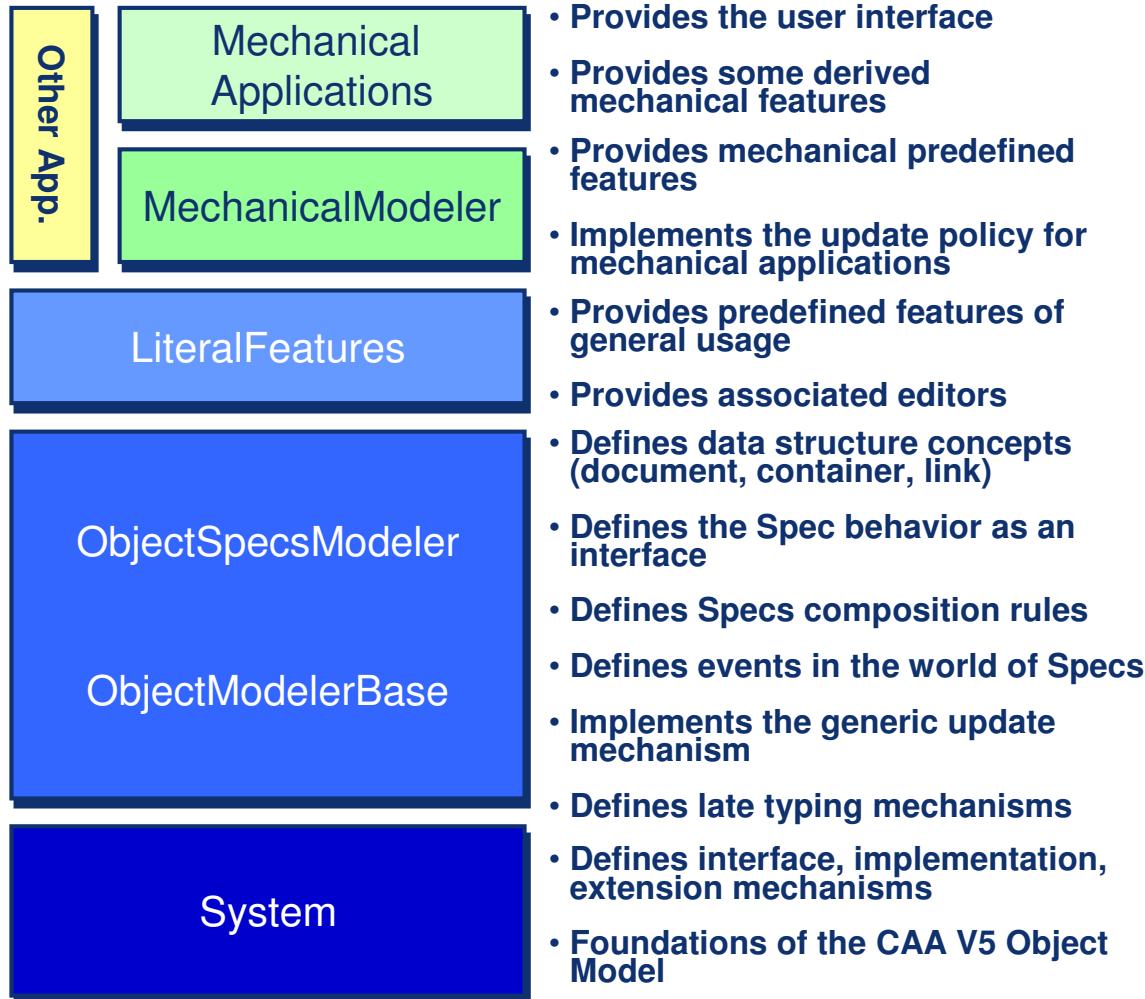
What applications can do

Use predefined mechanical features

Use predefined basic features

Define their own specs
Benefits from update mechanism

Use the CAA V5 Object model



Student Notes:

The CATPart Document Access (1/2)

CATDocumentServices global methods

- ◆ Save
- ◆ SaveAs
- ◆ Open
- ◆ New
- ◆ NewFrom
- ◆ Remove

```
CATDocument* pDocument = NULL;  
rc = CATDocumentServices::New("Part", pDocument);  
if ((FAILED(rc)) || (NULL == pDocument))  
{  
    cout << "ERROR in creating New document" << endl;  
    return 1;  
}  
else cout << "--> OK" << endl;
```

- ◆ All methods of this class must be used to create, open or close a document when **no visualization is necessary**. This is always the case in batch mode, but it is also possible in interactive session (see next slide).

Student Notes:

The CATPart Document Access (2/2)

CATIInIInteractiveSession

- ◆ Save
- ◆ SaveAs
- ◆ Open
- ◆ New
- ◆ NewFrom

```
CATSession *session = CATSession::GetPtrSession();
if (session){

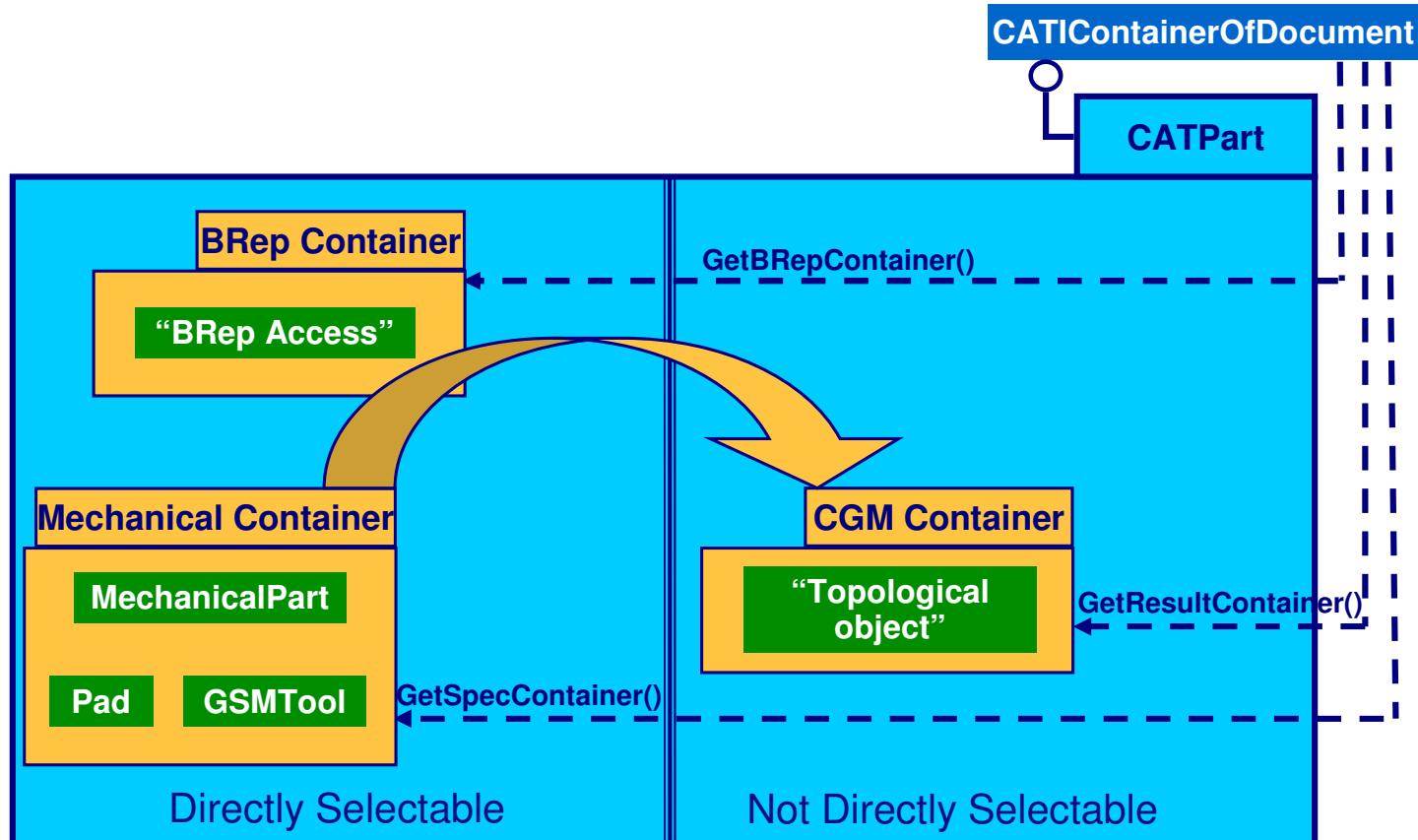
    CATIInIInteractiveSession *spII Session = NULL;
    hr = session->QueryInterface(IID_CATIInIInteractiveSession,
        (void **) &spII Session);

    if (SUCCEEDED(hr))
    {
        CATIEditor *piEdit = NULL;
        hr = spII Session-> New("Part", &piEdit);
        ...
    }
}
```

- If you want to visualize the document (in an interactive session), use the **CATIInIInteractiveSession** interface. This last interface is defined in the InteractiveInterfaces Framework.

[Student Notes:](#)

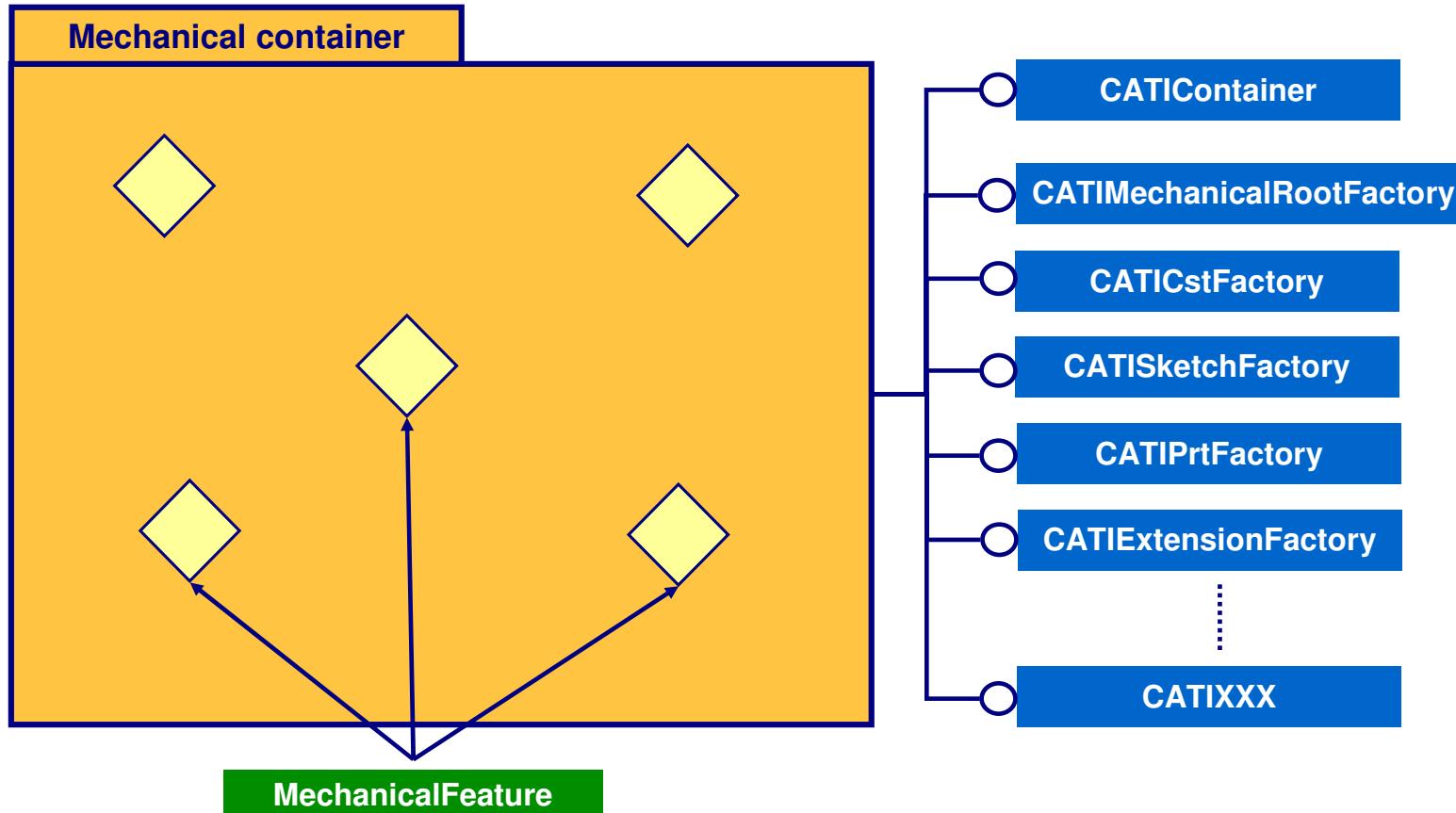
The CATPart Document Structure (1/2)



Student Notes:

The CATPart Document Structure (2/2)

- Mechanical feature container:



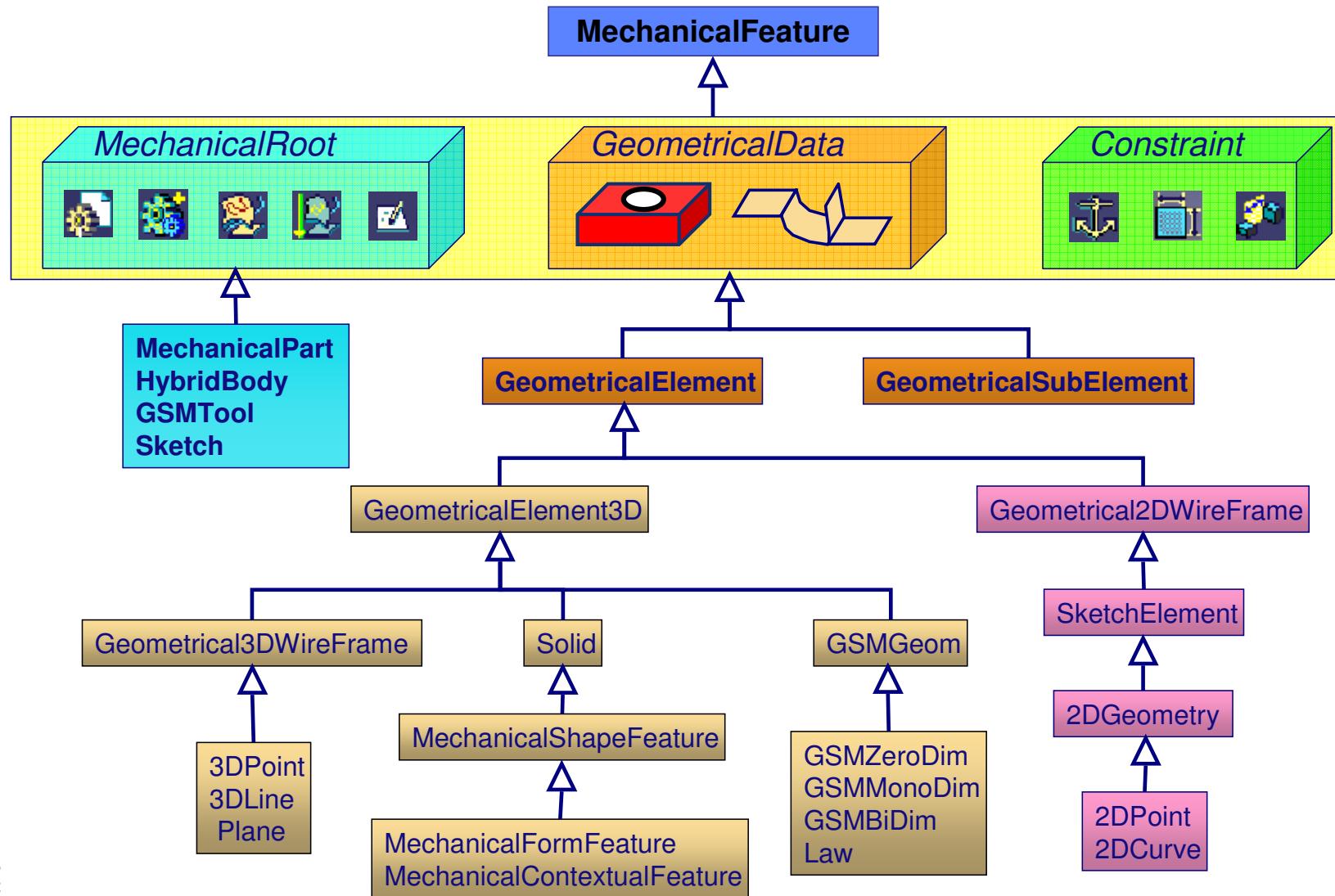
[Student Notes:](#)

Main Mechanical Modeler Feature Categories

- **MechanicalRoot:**
 - ◆ a feature aggregating other features
- **GeometricalElement:**
 - ◆ a feature generating a topological body as a result
- **GeometricalSubElement:**
 - ◆ a feature accessing in a stable way to a sub-element of the result of a Geometrical Element.
- **Constraint:**
 - ◆ a feature defining constraints between features

[Student Notes:](#)

Mechanical Modeler Feature Hierarchy



Student Notes:

CAA V5 Mechanical Modeler: The “Root” Features

This describes all root features involved in the CAA V5 Mechanical Modeler.

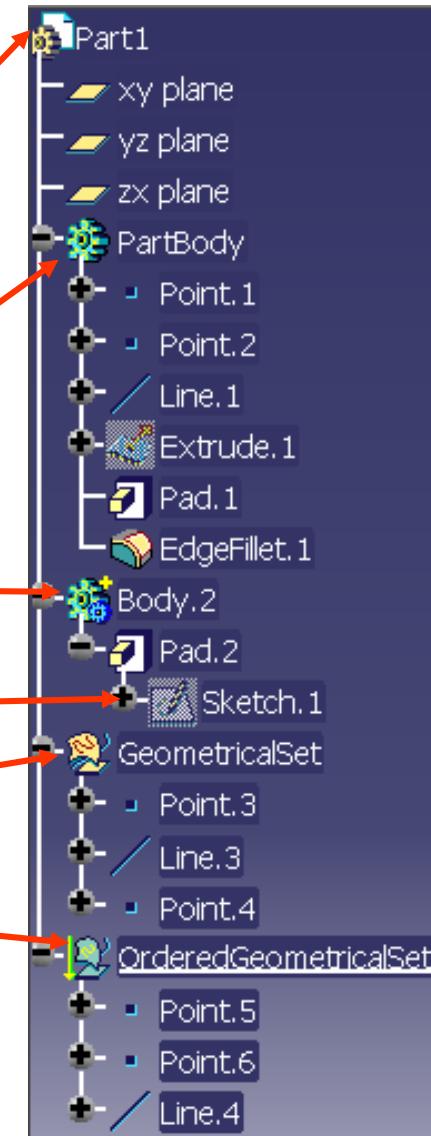
- The Root Features
- MechanicalPart Features
- Tool Features

[Student Notes:](#)

MechanicalRoot Features

- Mechanical features are aggregated in “Root” features.

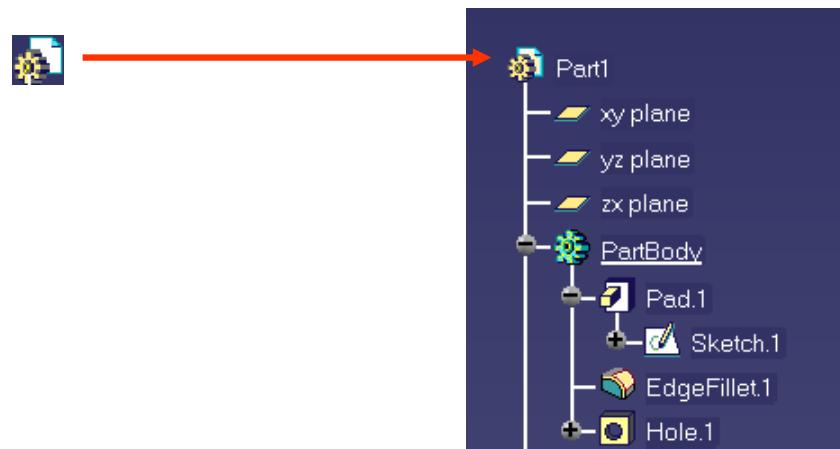
Root Features

MechanicalPart**HybridBody****Sketch****GSMTool**

Student Notes:

MechanicalPart Features

- It is the top level feature of the CATPart document. It aggregates directly or indirectly all the other mechanical features. It is unique.
- CATIPartRequest to retrieve all the Mechanical Bodies in the MechanicalPart



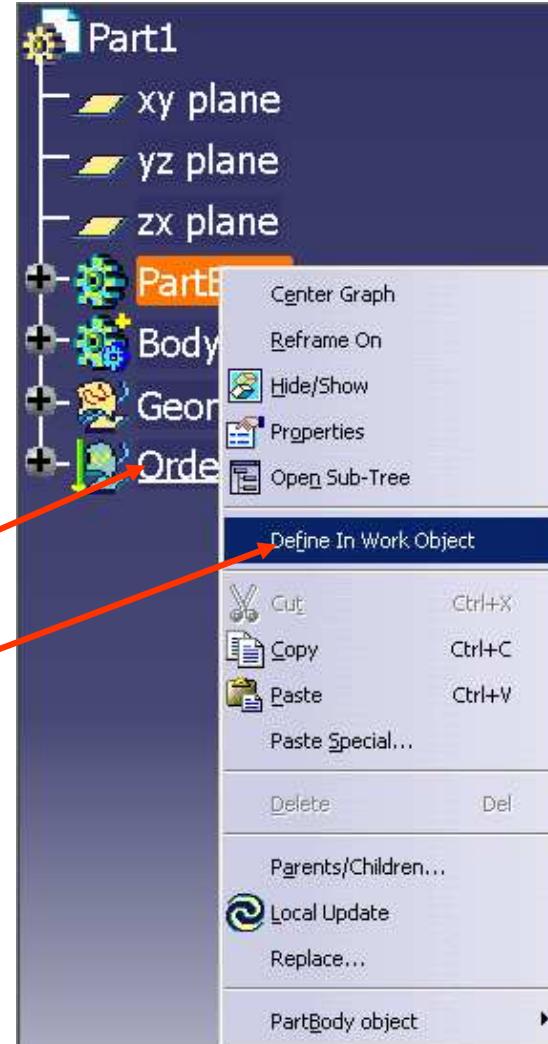
Student Notes:

What is a Tool?

- A tool is an aggregating feature. There are three main types:
 - ◆ HybridBody dedicated for all mechanical features (solid, surface and wireframe)
 - ◆ GSMTTool for surface and wireframe features
 - ◆ Sketch for 2D features
- Several tools can be aggregated in a part, but only one is active at a given time.

Active tool

Command to activate a tool

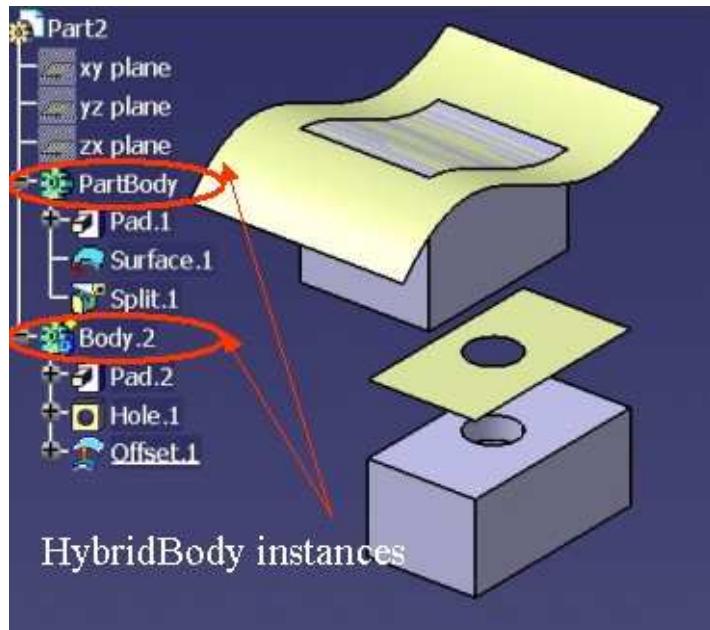


- You can change the activated feature with the **SetCurrentFeature()** method of the **CATIPrtPart** interface on a **MechanicalPart** feature.

[Student Notes:](#)

HybridBody Features

- HybridBody features contain solid, surface and wireframe features to simplify the hybrid design.
 - ◆ Before V5R14, PartBody was a MechanicalTool feature (containing only solid).
 - ◆ There will not be any conversions for the MechanicalTool feature (a MechanicalTool stored in V5R13 and open in V5R14 will be always a MechanicalTool).



- It is based on two main notions:
 - ◆ Order
 - ◆ Linearity

[Student Notes:](#)

GSMTTool Features

- ❖ GSMTTool feature aggregates wireframe or surface features.
- ❖ There are two kind of GSMTTool: Geometrical Set and Ordered Geometrical Set
- ❖ Geometrical Set:
 - ◆ Adapted for features with few history or without input specification
 - ◆ Parent-Children management

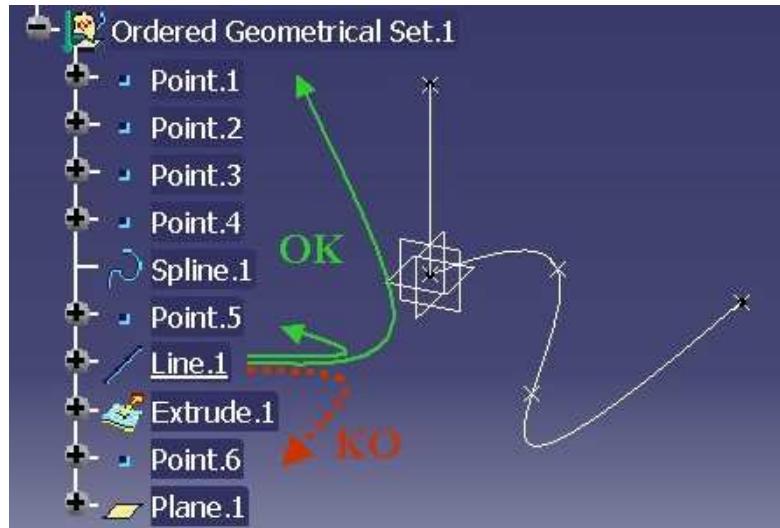


- ❖ Ordered Geometrical Set:
 - ◆ enables you to simplify the links management between features when the Part has a huge historical graph.
 - ◆ It is based on two main notions: Order and Linearity

[Student Notes:](#)

Order Principle

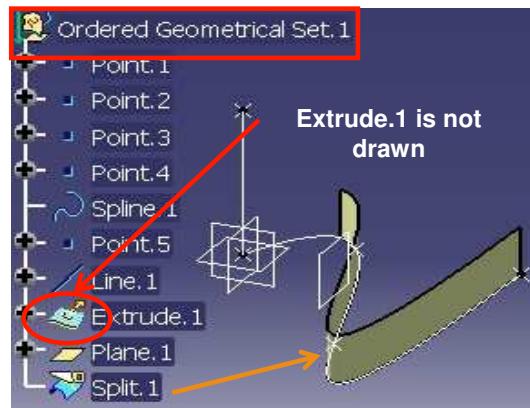
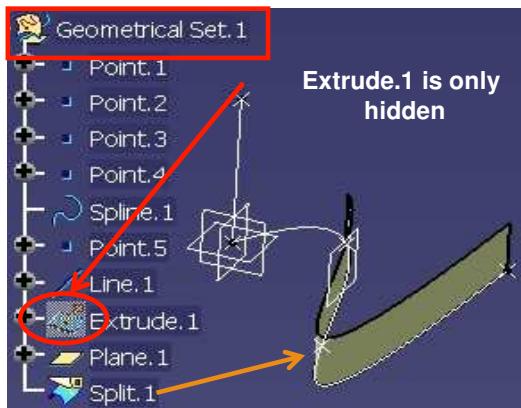
- The goal is to have a body where features are ordered from the basic features (top) to the features with huge historical graph (bottom).
- Each feature inside an ordered body can only have as input specifications features which are above.



Student Notes:

Linearity Principle

- ◆ The goal is to have a body where only relevant features are taken into account. Those specific features are named **absorbent** features.
- ◆ Absorbent features modify other features. Their visualization makes the visualization of the modified features useless (**absorbed** features), since they are superimposed.
- ◆ Ex: The Split feature, is an absorbent feature. The split surface (ex: an Extrude feature) is an absorbed feature



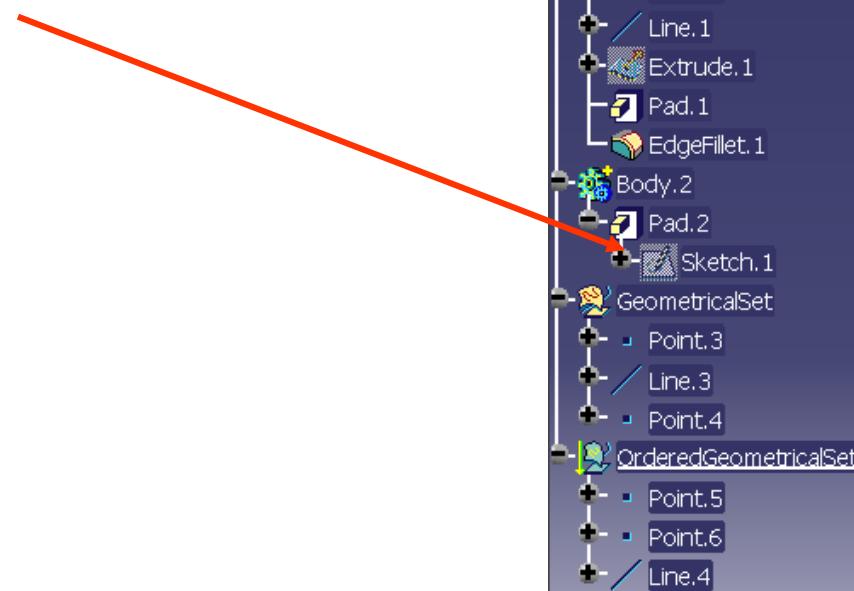
- ◆ CATIISystemDescription enables you to define the type of the surfacic feature.

[Student Notes:](#)

The Sketcher Tool

- Sketch feature aggregates 2D point, wireframe and 2D constraints. It is displayed in the specification tree under the default name Sketch.

Sketcher Tool

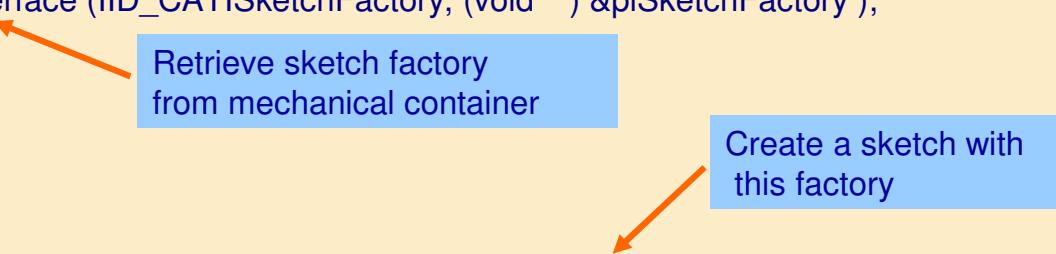


Student Notes:

The Sketch object creation (1/2)

- As in interactive, you need to create a sketch with the factory

```
CATISketchFactory * piSketchFactory =NULL;  
hr = spSpecCont->QueryInterface (IID_CATISketchFactory, (void **) &piSketchFactory );  
... // Test result  
  
double origin[3] = {0,0,0};  
double xAxis[3] = {1,0,0};  
double yAxis[3] = {0,1,0};  
double zAxis[3] = {0,0,1};  
CATISpecObject_var spSpecOnSketch = piSketchFactory->CreateSketch(origin, xAxis, yAxis);  
... // Test result
```



- As in interactive, you need to open an edition before creating 2D elements (and constraints) and closing the edition

```
CATISketch * piSketch = NULL;  
rc = spSpecOnSketch->QueryInterface (IID_CATISketch, (void**) &piSketch);  
... // Test result  
  
// Open a edition  
piSketch->OpenEdition();
```

Student Notes:

The Sketcher object creation (2/2)

■ Create 2D elements (and constraints if needed)

```
CATI2DWFFactory * piFactoryOnSketch= NULL  
hr = spSpecOnSketch->QueryInterface (IID_CATI2DWFFactory, (void**) &piFactoryOnSketch);  
... // Test result
```

Retrieve the 2D wireframe
factory from sketch feature

Creation of 2D wireframe
With this factory

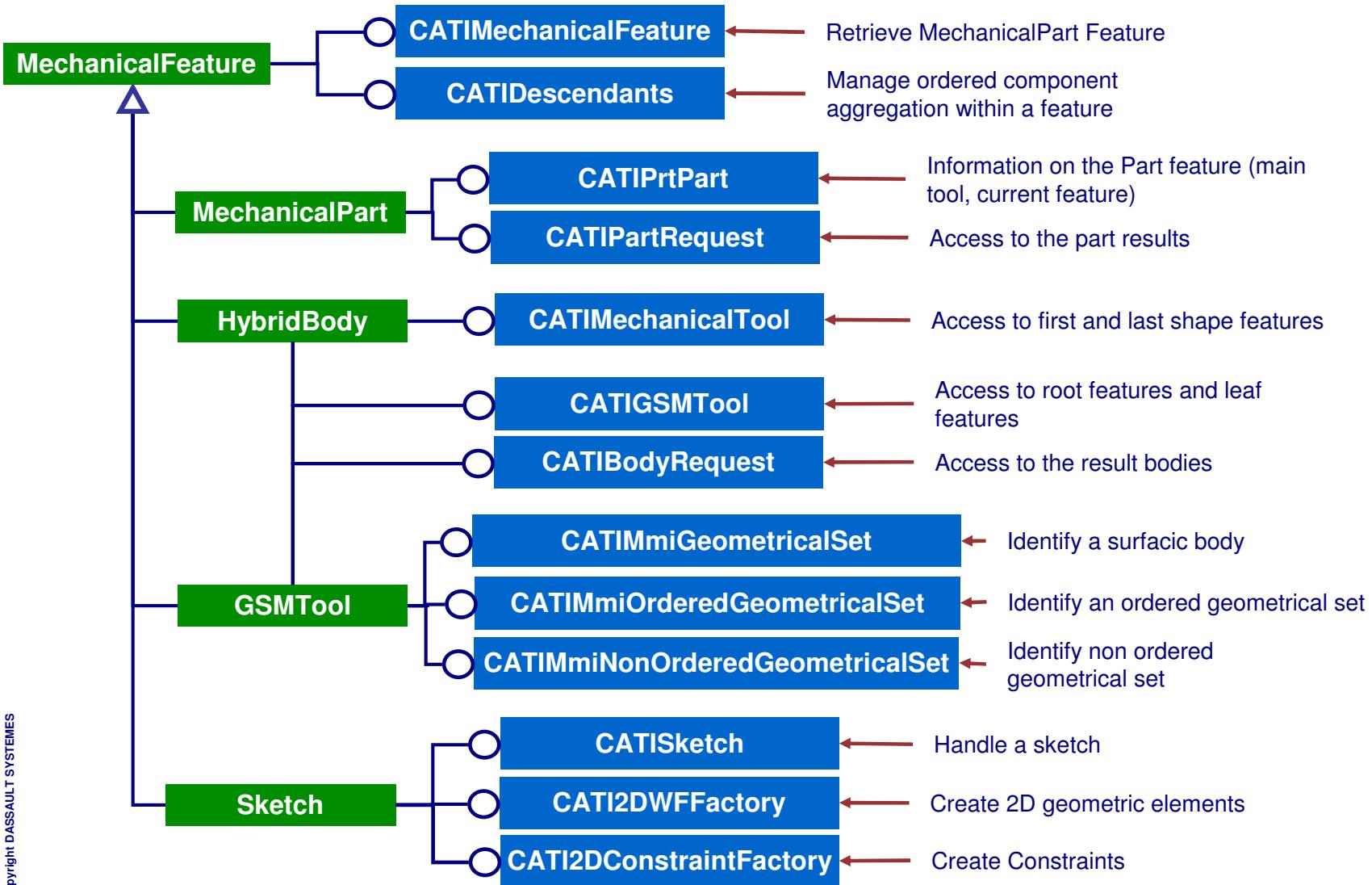
```
CATISpecObject_var spSketchPoint = piFactoryOnSketch->CreatePoint(...);  
CATISpecObject_var spSketchLine = piFactoryOnSketch->CreateLine(...);
```

■ Close the edition

```
piSketch->CloseEdition();
```

Student Notes:

Main Interface for the “Root” Features



[Student Notes:](#)

CAA V5 Mechanical Modeler: The GeometricalElement3D Features

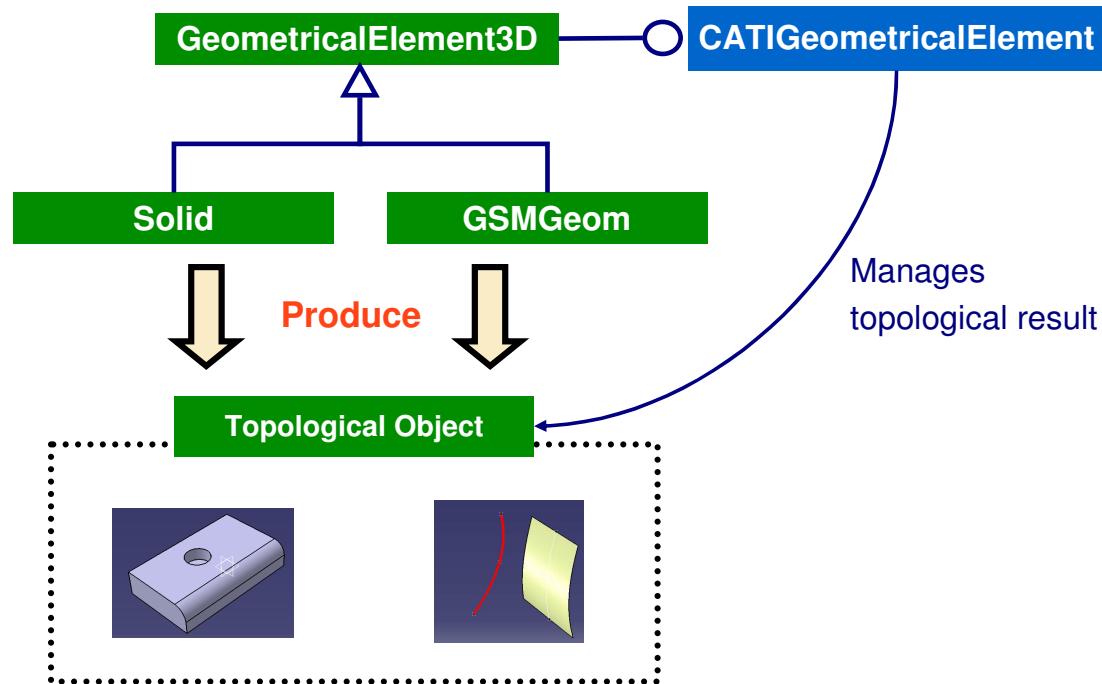
This describes all the geometrical features involved in the CAA V5 Mechanical Modeler.

- Topological Link
- Solid Feature
- Mechanical Form Feature
- Mechanical Contextual Feature
- GSMGeom Feature

Student Notes:

Link with Topology and Geometry

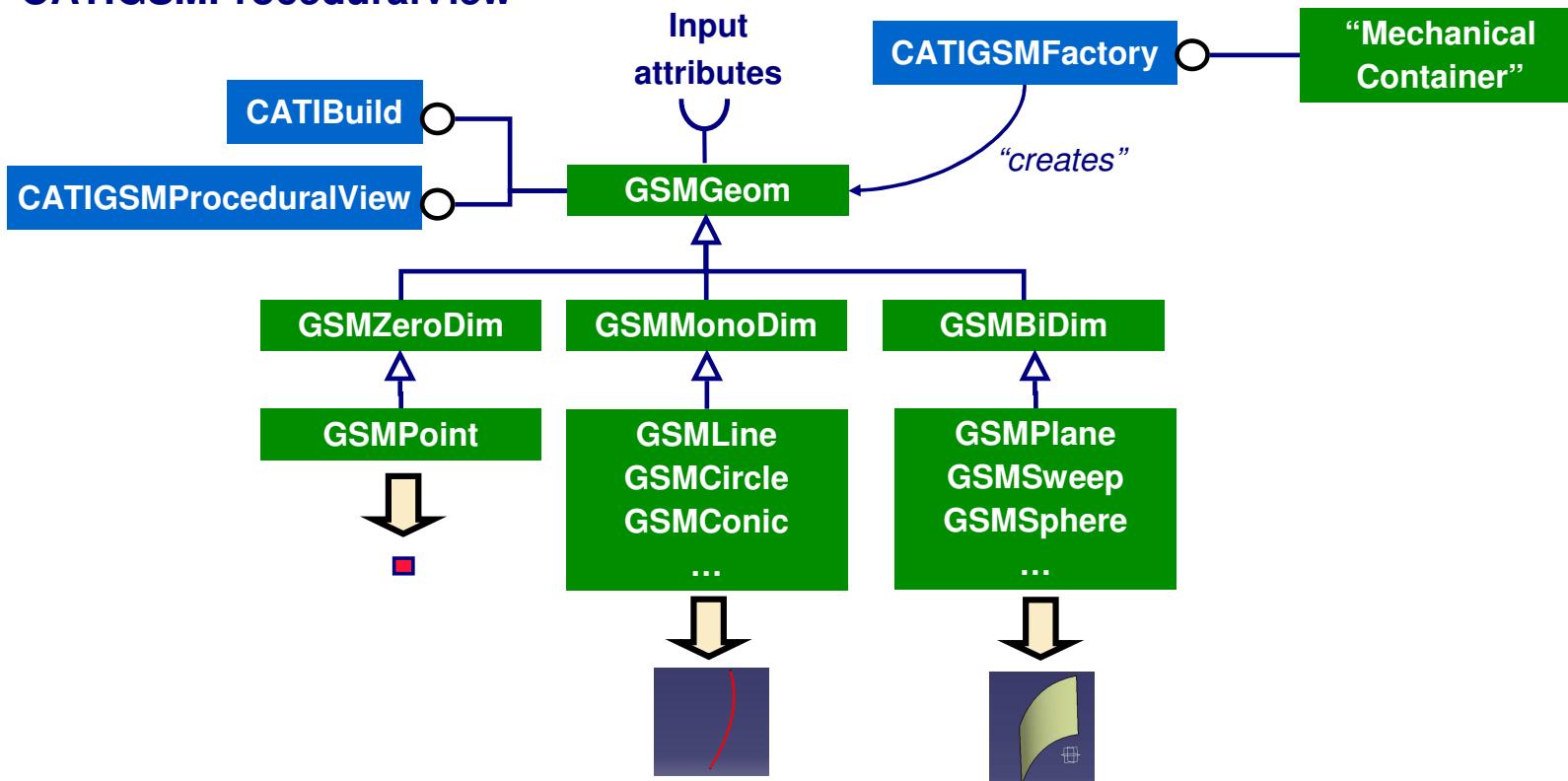
- A 3DGeometricalElement feature generates a topological result that is a body.
- To retrieve the topological body, use the method GetBodyResult() from the CATIGeometricalElement interface.



Student Notes:

The GSMGeom features

- Base Feature for surfacic and wireframe feature inserted in a HybridBody or a GSMTool
 - CATIGSMFactory allow you to create DS surfacic or wireframe features
 - After creating, you have to aggregate it under a tool thanks to CATIGSMPProceduralView



Creating Wireframe and surface Features example:

[...]

```
CATICkeParmFactory_var spCkeParmFact= spSpecCont;  
  
double XCoord = 0.;  
double YCoord = 0.;  
double ZCoord = 1.;  
CATICkeParm_var spFirstCoord      = spCkeParmFact ->CreateLength( "X", XCoord);  
CATICkeParm_var spSecondCoord    = spCkeParmFact ->CreateLength( "Y", YCoord);  
CATICkeParm_var spThirdCoord     = spCkeParmFact ->CreateLength( "Z", ZCoord);
```

Use KW Factory to
create X, Y and Z
parameters

Creation of a GSMPPoint

```
CATIGSMFactory_var spGSMFactory = spSpecCont;  
CATIGSMPPointCoord_var spGSMPPoint = spGSMFactory-> CreatePoint(spFirstCoord,  
                           spSecondCoord,  
                           spThirdCoord);
```

```
CATISpecObject_var spSOonGSMPPoint = spGSMPPoint;  
spSOonGSMPPoint->Update();
```

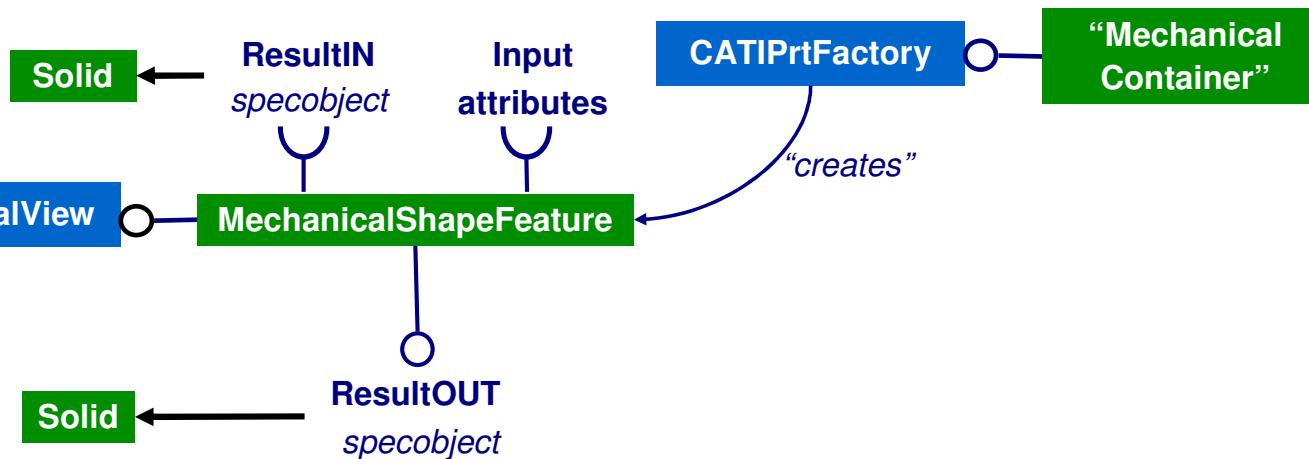
[...]

You must **update** your feature.

Student Notes:

The Solid Feature (1/2)

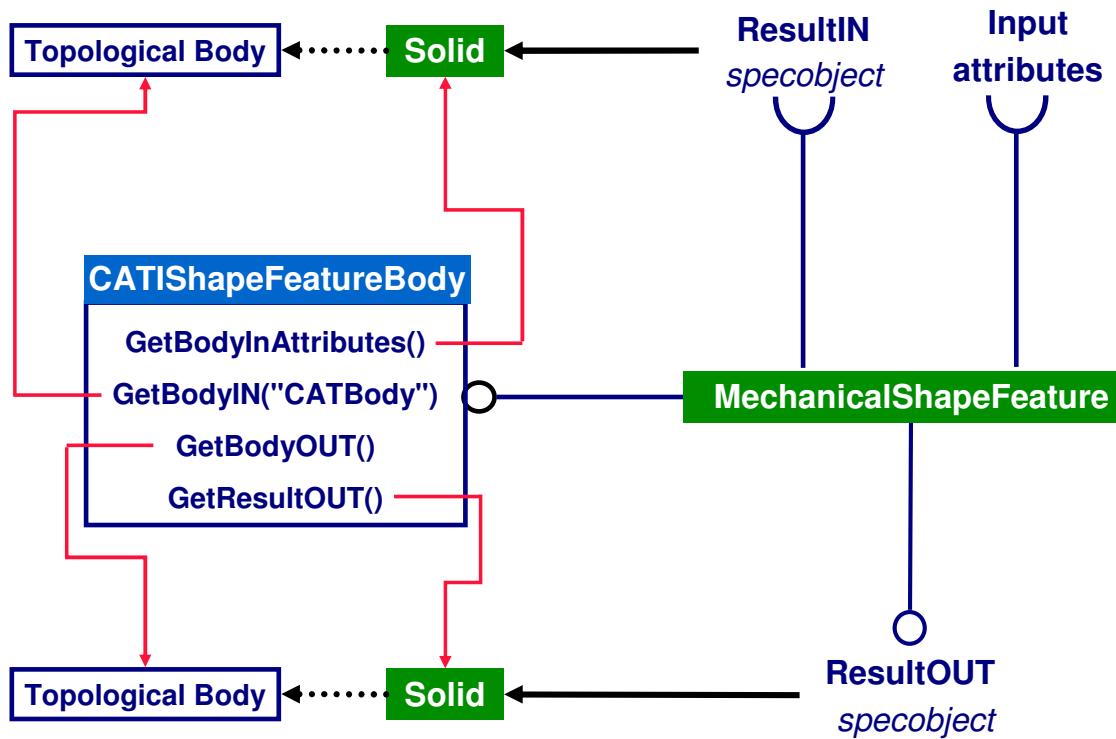
- ResultIN references an input Solid feature and ResultOUT aggregates an output Solid feature.
- CATIPrtFactory allow you to create DS solid feature
- After creating, you have to aggregate it under a hybridbody thanks to CATIPrtProceduralView



[Student Notes:](#)

The Solid feature (2/2)

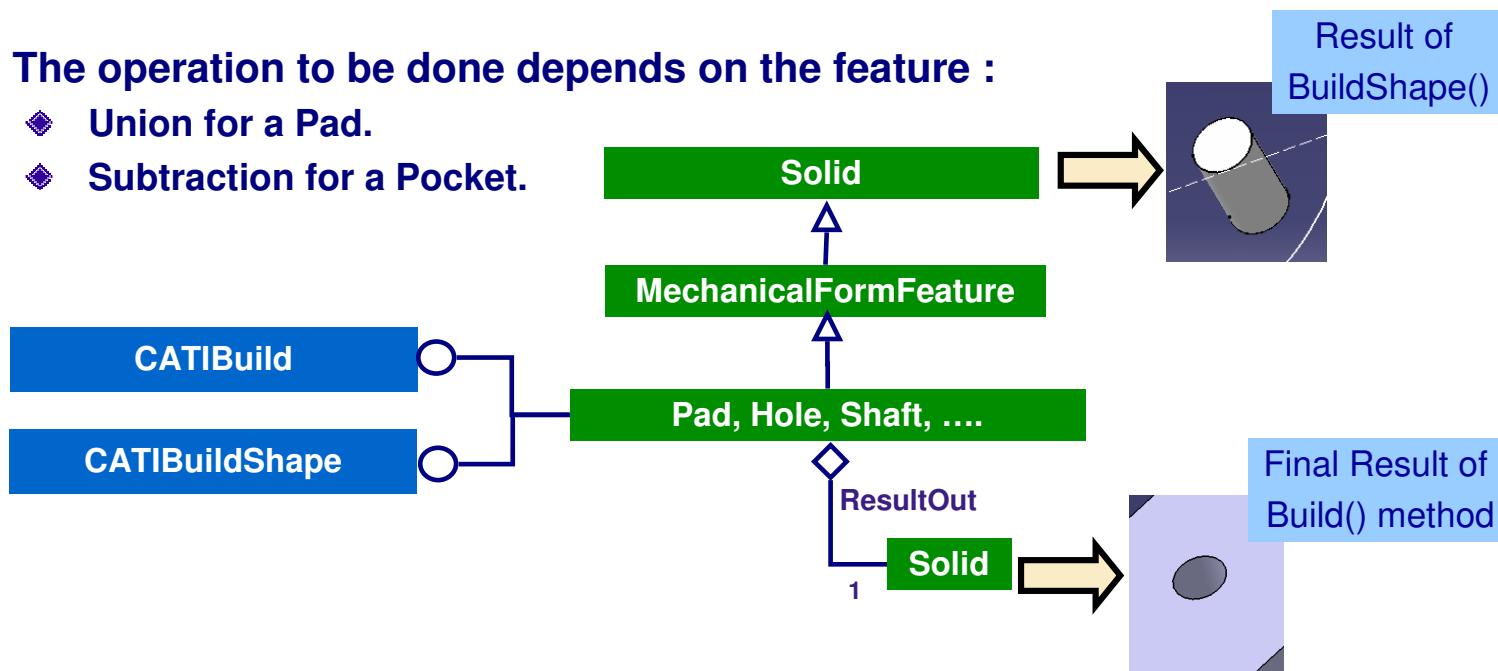
- Use the `CATIShapeFeatureBody` interface to retrieve the `ResultIN` and `ResultOUT` feature and the corresponding Topological body.



Student Notes:

The MechanicalFormFeature feature

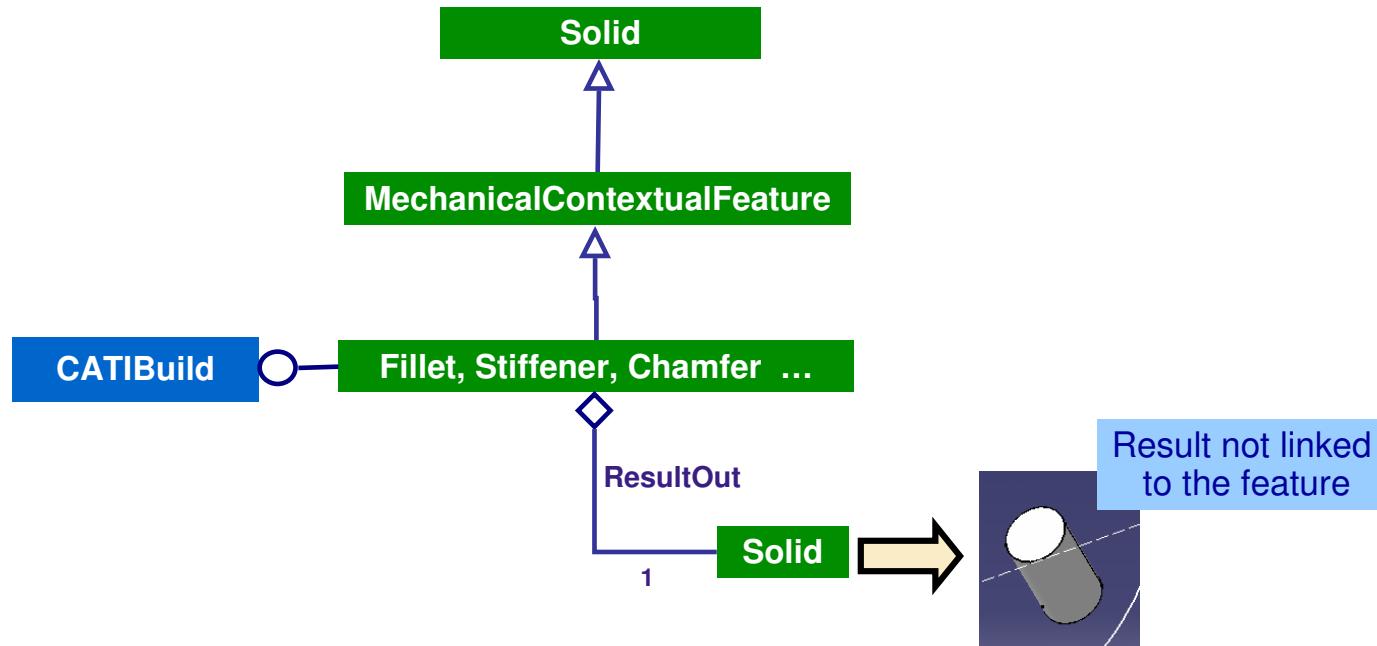
- Creates material by itself, even without context.
- It implements the BuildShape method called in the Build method
- The Build method calls first BuildShape to build the shape of the form feature, then operates the previous shape with the resulting solid of the previous feature in the procedural view.
- The operation to be done depends on the feature :
 - ◆ Union for a Pad.
 - ◆ Subtraction for a Pocket.



Student Notes:

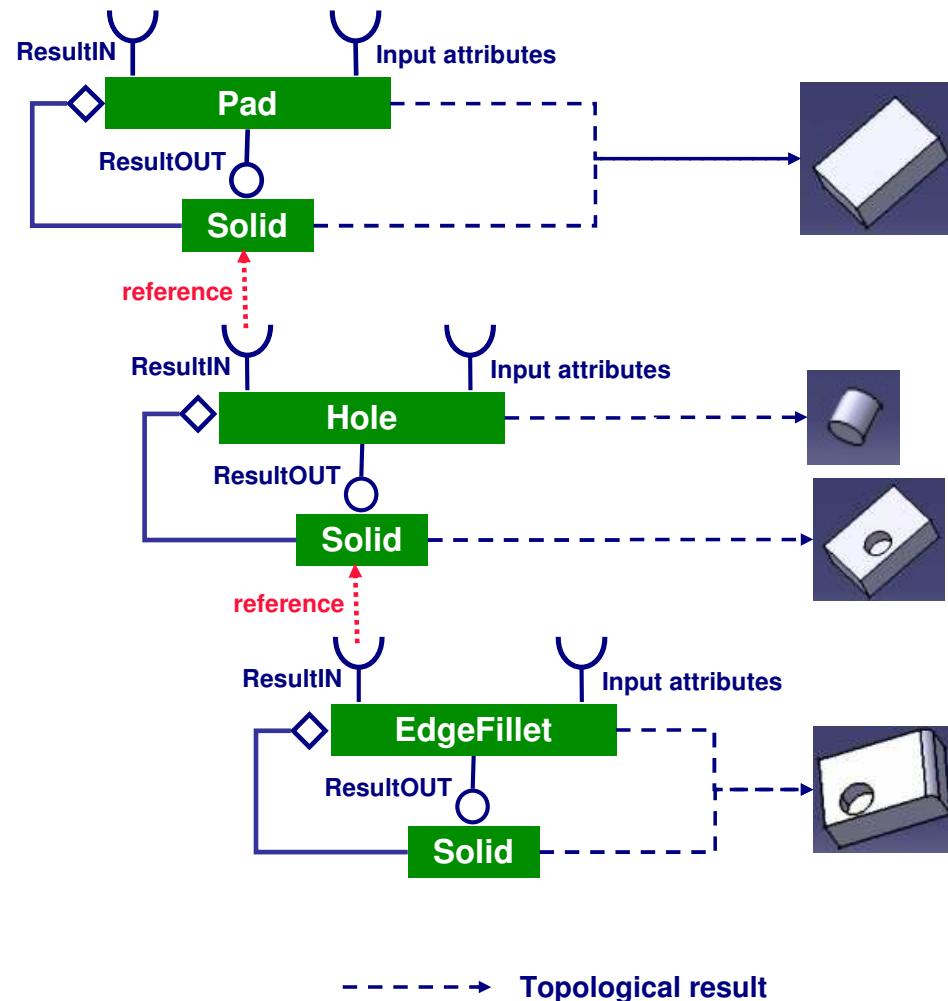
The MechanicalContextualFeature features

- No BuildShape method



Student Notes:

Solid architecture sample



[Student Notes:](#)

Creating Solid feature example:

[...]

```
// a - Retrieve Sketcher as SpecObject
CATISpecObject_var spSpecOnSelectedSketch= spSketch;
double HeadHeight = 11;

// b – Create the Pad
CATIPrtFactory_var spPrtFactory = spSpecCont;
CATISpecObject_var spSpecOnPad = spPrtFactory ->CreatePad(spSpecOnSelectedSketch);

// c - Modify Pad attributes
CATIPad_var spPad = spSpecOnPad;
spPad->ModifyEndOffset(HeadHeight);
spPad->ReverseDirection();

spSpecOnPad->Update();
```

Retrieve CATISpecObject Interface on the sketch feature

Modify Pad attributes

Modify Pad attributes

You must **update** your feature.

[...]

Student Notes:

CAA V5 Mechanical Modeler: The “Constraint” Features

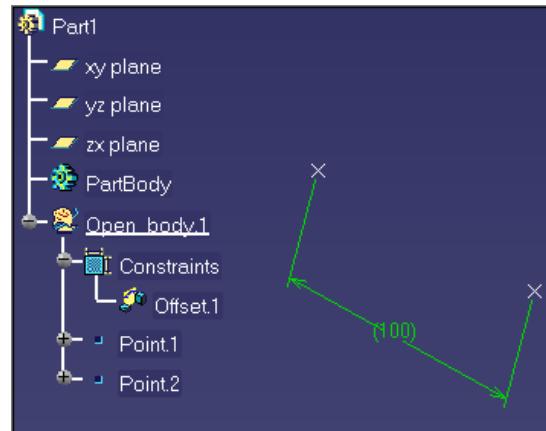
This describes all Constraint features involved in the CAA V5 Mechanical Modeler.

- The “Constraint” Features
- Interfaces to manipulate “Constraint” Features
- Example

Student Notes:

The “Constraint” Features

- ❖ The Constraint features enable constraint definition between one or several geometrical features.
- ❖ A constraint is defined by:
 - ◆ its type: distance / symmetry / ...
 - ◆ its mode: measured or constraining
 - ◆ its configuration : angle sector, orientation...
- ❖ It applies to up to three elements.
 - ◆ fix, radius/diameter (1)
 - ◆ Surface Contact (2)
 - ◆ Symmetry (3 elements)
- ❖ It may have a value or not.
 - ◆ Offset
 - ◆ Parallelism
- ❖ Available in Sketcher (on 2D elements), Part Design, Assembly Design...



Student Notes:

Interface Summary for the “Constraint” Features

- Standard DS Constraints (Part / Assembly / Sketcher) deriving from Constraint are used through the CATICst interface.
- Constraints are created through CATICstFactory.

[...]

```
CATICstFactory_var spCstFactory = spSpecCont;  
  
CATCstType iType = CstType_Distance ;  
CATBaseUnknown_var spFirstPointFeature = spFirstSelectedFeature;  
CATBaseUnknown_var spSecondPointFeature = spSecondSelectedFeature;  
double iValue = 15;  
  
CATICst_var spCst = spCstFactory ->CreateStdConstraint ( iType,  
                                         spFirstPointFeature,  
                                         spSecondPointFeature,  
                                         iValue);
```

[...]

Student Notes:

CAA V5 Mechanical Modeler: The “Axis System” Features

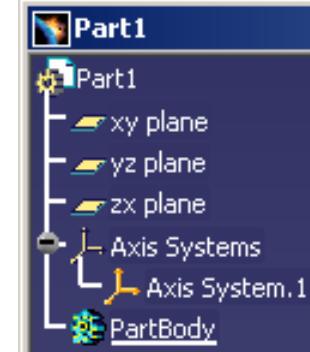
This describes special axis system features involved in the CAA V5 Mechanical Modeler.

- The “Axis System” Features
- Example

Student Notes:

The “Axis System” Features

- Several axis systems can be defined in a part document
- Different kind of Axis System :
 - ◆ Standard ,
 - ◆ Axis Rotation,
 - ◆ Euler Angle...
- An axis system can be set as current (it is highlighted)
- Axis Systems are used through the **CATIMf3DAxisSystem** interface.
- Axis Systems are created through **CATIMf3DAxisSystemFactory** interface.



Creating an Axis System - Example

Student Notes:

```
CATIMf3DAxisSystemFactory * pIMf3DAxisSystemFactory = NULL ;  
oRC = spSpecCont->QueryInterface(IID_CATIMf3DAxisSystemFactory,  
(void **) & pIMf3DAxisSystemFactory);  
  
if ((SUCCEEDED(oRC)) && (NULL != pIMf3DAxisSystemFactoryOnFeatCont))  
{  
    CATIMf3DAxisSystem_var spAxisSystem;  
    CATMathPoint Origin (150.0,.0,.0);  
    CATMathVector X (1.0,.0,.0);  
    CATMathVector Y (0.0,1.0,.0);  
  
    oRC = pIMf3DAxisSystemFactory->CreateAxisSystem(Origin,X,Y, spAxisSystem);  
  
    pIMf3DAxisSystemFactoryOnFeatCont->Release();  
    pIMf3DAxisSystemFactoryOnFeatCont = NULL;  
}
```

Retrieve Axis System factory
from the mechanical container

Create Axis System from
origin + directions

Exercise Presentation

→ Screw : First part

And now practice on 2 exercises, to learn about:

- ◆ Using a Batch to create an empty CATPart document :
 - Creating a new workspace
 - Creating a new batch program

- ◆ Creating DS object (Screw) in batch mode :

- Using DS factories
 - Creating Sketch and solids

Student Notes:

Student Notes:

CAA V5 Drafting and Tolerancing

In this lesson you will have an overview of the capabilities offered in Drafting and FTA in CAA.

- **Drafting Overview**
- **Drafting objects and interfaces**
- **FTA overview**
- **FTA objects and interfaces**

[Student Notes:](#)

Drafting Overview

- The Drafting component offers a library of objects to create or modify:
 - ◆ Geometric elements.
 - ◆ Annotation elements.
 - ◆ Structuring objects (Sheet, view, ditto).

- These components are to be used “as is”
 - ◆ because User Objects would not inherit these mechanisms.

[Student Notes:](#)

Drafting : Managing Sheets and Views (1/2)

- ❖ A drawing is a collection of sheets
- ❖ The sheet corresponds to the paper space, where the following elements are positioned.
 - ◆ One main view that supports the geometries directly created in the sheet.
 - ◆ One background view dedicated to frames, title blocks...
 - ◆ Any number of Interactive and Generative views.
- ❖ A View may contain
 - ◆ 2D Geometry
 - Geometry created by using the Drafting workbench.
 - Sketcher component is used to create 2D geometry.
 - ◆ Generative results:
 - Non modifiable 2D geometry created from 3D document and linked on it.
 - ◆ Constraints
 - Sketcher component is used to create Constraints.
 - ◆ Annotations and Dress-up
 - Dimensions, texts, arrows, area fill, axis line, ...

Student Notes:

Drafting : Managing Sheets and Views (2/2)

```
CATDocument* pDocDrawing = NULL;
CATDocumentServices::OpenDocument (pfileNameDrawing, pDocDrawing);
```

Open drawing document

```
CATIDftDocumentServices *piDftDocServ = NULL;
pDocDrawing->QueryInterface (IID_CATIDftDocumentServices, (void **)&piDftDocServ);
CATIDftDrawing *piDrawing = NULL;
piDftDocServices-> GetDrawing (IID_CATIDftDrawing, (void **)&piDrawing);
```

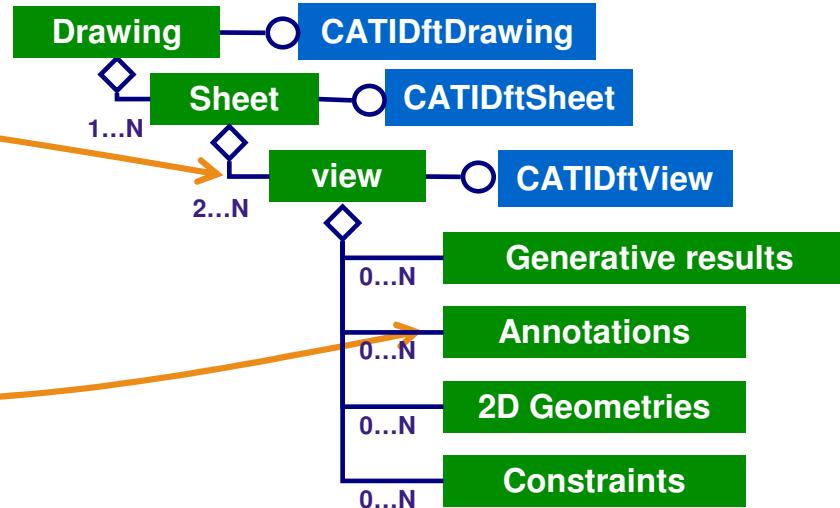
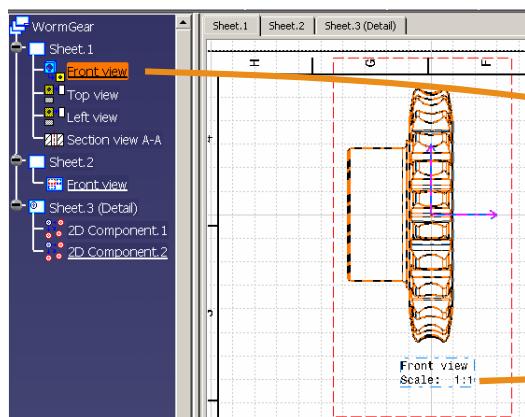
Get pointer on drawing root

```
CATIDftView *piDftView = NULL;
piDrawing->GetActiveView (&piDftView);
```

Get active view of active sheet

```
CATIUnknownList * piListOfTexts = NULL;
piDftView->GetComponents (IID_CATIDftText,&piListOfText);
```

Get all texts in this view



Student Notes:

Drafting : Creating Geometry

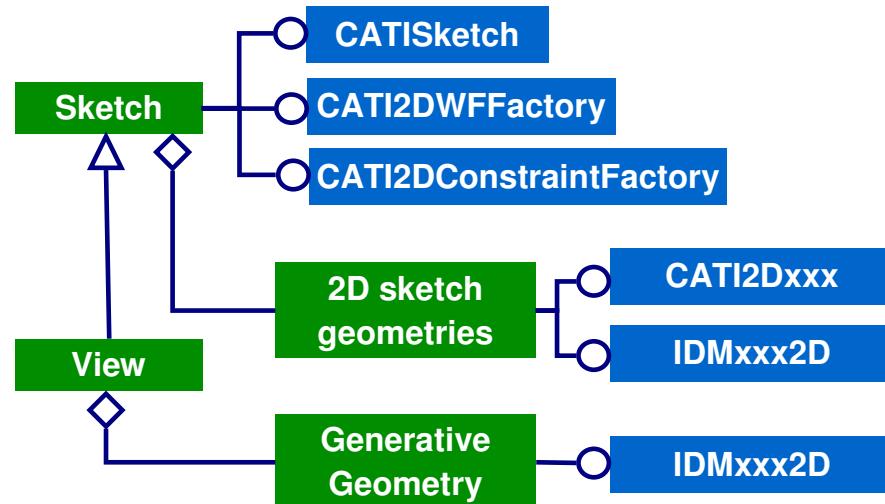
- Two kinds of geometry coexist in a View:
 - 2D geometry managed by the Sketcher component.
 - Generative geometry created from extraction of 3D documents.

```
CATI2DWFFactory *piGeomFactory = NULL;
piDftView->QueryInterface(IID_CATI2DWFFactory, (void **)&piGeomFact);
```

```
double startPoint={20.,30.}; double endPoint={20.,70.};
CATISpecObject_var spLine;
spLine = piGeomFact->CreateLine (startPoint, endPoint);
```

Get Wireframe
factory

Create Geometric
objects



Student Notes:

Drafting : Additionnal information

- A new toolbar can be inserted in Drafting Workbench by implementing **CATIDrwAddin** interface.
- Drafting provides a factory for annotation objects : **CATIDrwAnnotationFactory**.
- Interface **CATIDrwGenDrawShape** allows to query only Generative geometry in a view.

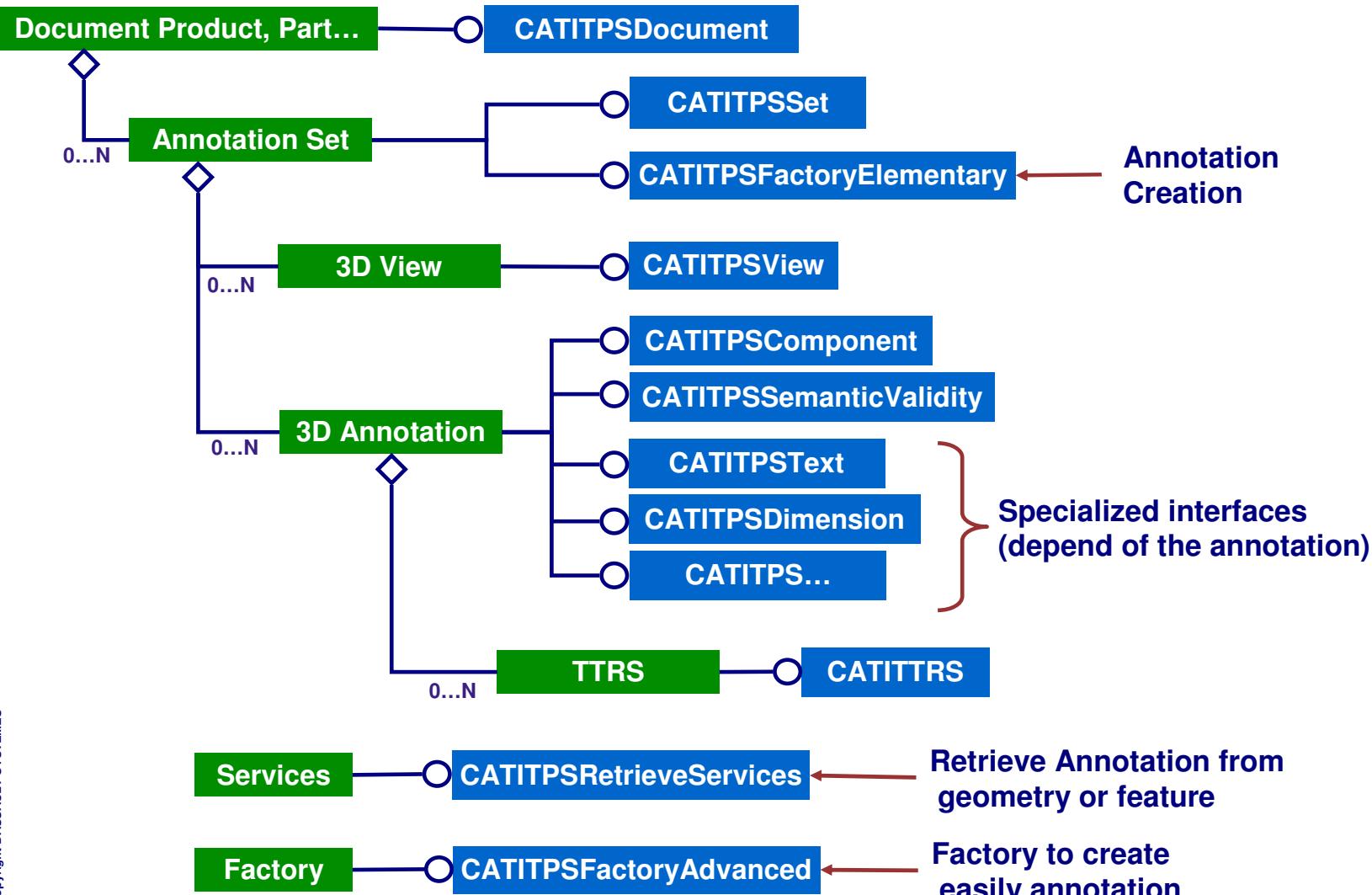
[Student Notes:](#)

Tolerancing Overview

- The Tolerancing component offers a library of objects to browse and create:
 - ◆ *Annotation Set.*
 - ◆ *3D Views.*
 - ◆ *FTA Captures.*
 - ◆ *User Surface and Group of User Surfaces (TTRS) creation.*
 - ◆ *Text, Flag Note, Non Semantic Dimension, Roughness, Note Object Attribute.*
 - ◆ *Datum feature, Datum target, Datum Reference Frame, Geometrical Tolerance, Semantic Dimension.*
- These components are to be used “as is”
- Note :
TTRS (Technologically and Topologically Related Surfaces) is a component provided by the MechModInterfaces framework used to link TPS with surfaces.

Student Notes:

Tolerancing : Object Model



Student Notes:

Tolerancing : Basic Creation

- Basic creation is done only through one method

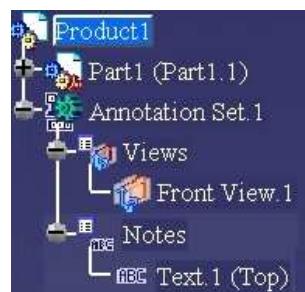
```
CATSO * pSelection = _pAgentGeometry -> GetListOfValues();

CATITPSFactoryAdvanced * piFactAdv = NULL;
rc = CATTPSInstantiateComponent (DfTPS_ItfTPSFactoryAdvanced, (void**) &
piFactAdv);

CATITPSText * piText = NULL;
CATUnicodeString TextString("Top");
CATMathPlane Plane = CATMathOIJ;
rc = piFactAdv -> CreateTextOnGeometry (pSelection, &Plane, &TextString, &piText);
```

Retrieve
CATITPSFactoryAdvanced
interface

Create the 3D text



Student Notes:

Tolerancing : Complex Creation (1/2)

Step 1 : Annotation set creation

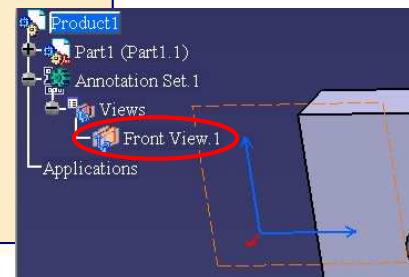
```
CATITPSServicesContainers * piTPSServicesCont = NULL;
CATITPSSet * piSetCopy = NULL;
CATBoolean bCreation = FALSE;
piTPSServicesCont -> RetrieveOrCreateCurrentTPSSet (piProduct, CreateIfMissing,
&piSetCopy, &bCreation);
```



Step 2 : View creation

```
CATMathPlane ViewPlane = CATMathOIJ;
CATITPSViewFactory * piViewFact = NULL;
piSet -> QueryInterface (IID_CATITPSViewFactory, (void**) &piViewFact);

CATITPSView * piTPSView = NULL;
piViewFact -> CreateView (&piTPSView, &ViewPlane, DftFrontView);
```



Student Notes:

Tolerancing : Complex Creation (2/2)

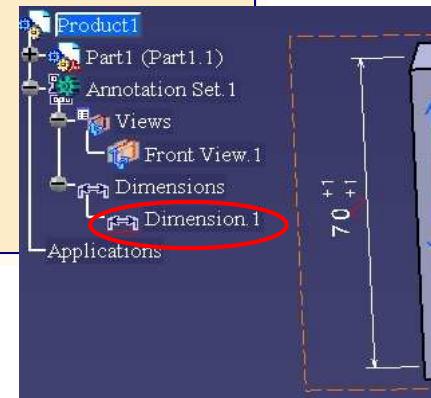
Step 3 : TTRS Creation

```
CATITPSFactoryTTRS * piTPSTTRSFact = NULL;
CATTPSInstantiateComponent (DfTPS_IItTPSFactoryTTRS, (void**)&piTPSTTRSFact);
CATMmrTTRSType CreationMode = CATMmrNodeTTRS;
CATSO pSO;
CATBaseUnknown * piBaseTop = NULL;
piPathTop -> QueryInterface (IID_CATBaseUnknown, (void**)&piBaseTop);
pSO.AddElement(piBaseTop); ...
pSO.AddElement(piBaseBottom);
CATITRS * piTTRS = NULL;
CATIPProduct * piProd = ...; // Product where you want to put 3D annotation
piTPSTTRSFact -> GetTTRS (&pSO, &piTTRS, CreationMode, piProd);
```

Step 4 : Create dimension

```
CATITPSFactoryElementary * piTPSFactElem = NULL;
piTPSSet -> QueryInterface (IID_CATITPSFactoryElementary, (void**)&piTPSFactElem);

CATITPSDimension * piTPSDim = NULL;
piTPSFactElem -> CreateSemanticDimension (piTTRS,
CATTPSLinearDimension,
CATTPSDistanceDimension,
&piTPSDim);
```



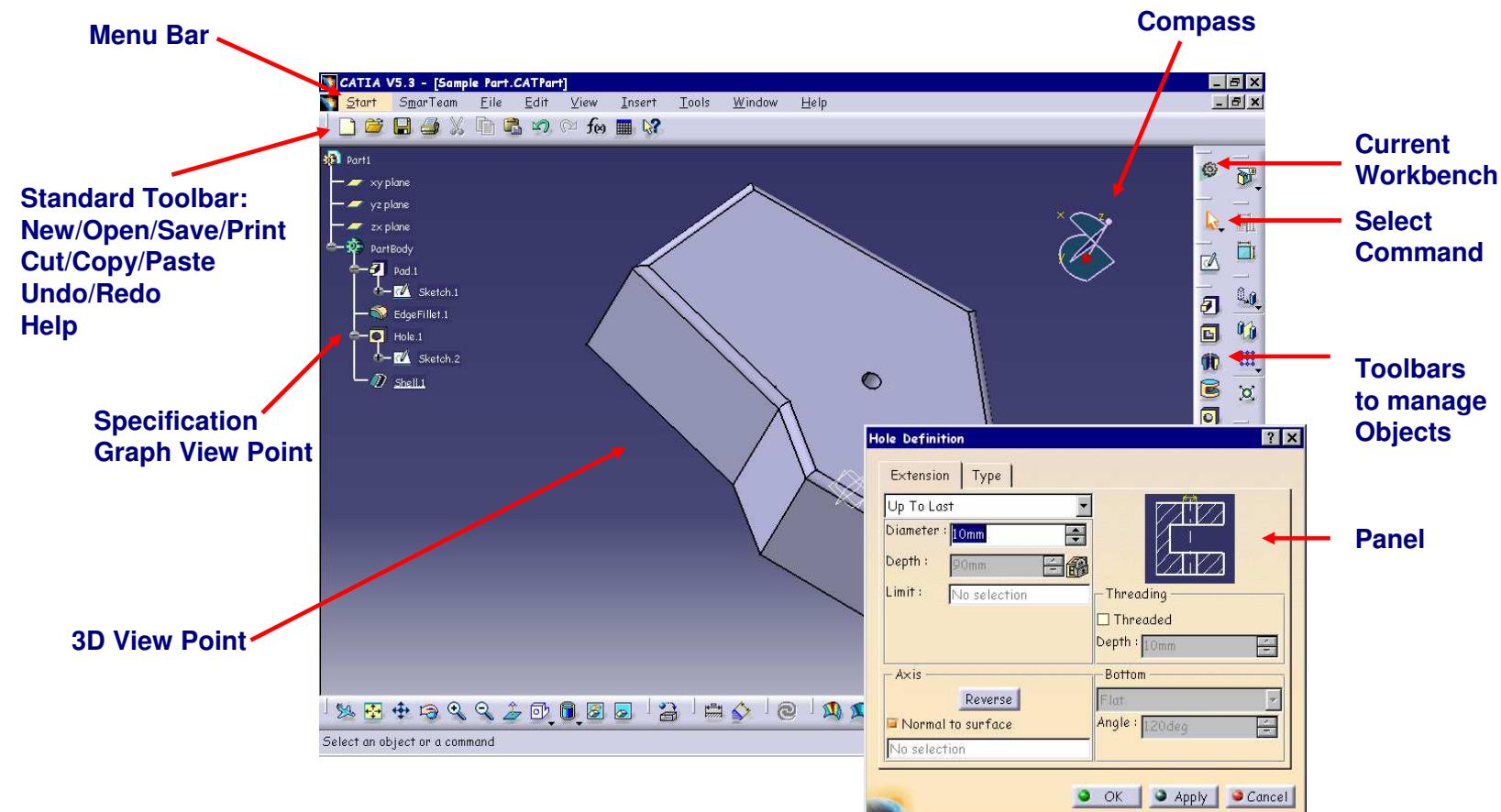
[Student Notes:](#)

CAA V5 ApplicationFrame

In this lesson you will learn the CATIA V5 frame editor, the Document Edition infrastructure, the main user interface principles and also how to plug your interactive application in CATIA.

- CATIA V5 frame editor
- Workshop, Workbench, Addin
- Integrating your application in CATIA
- Retrieve a document from the editor

Look and Feel of the CATIA V5 frame editor



Student Notes:

Frame editor class: CATFrmEditor

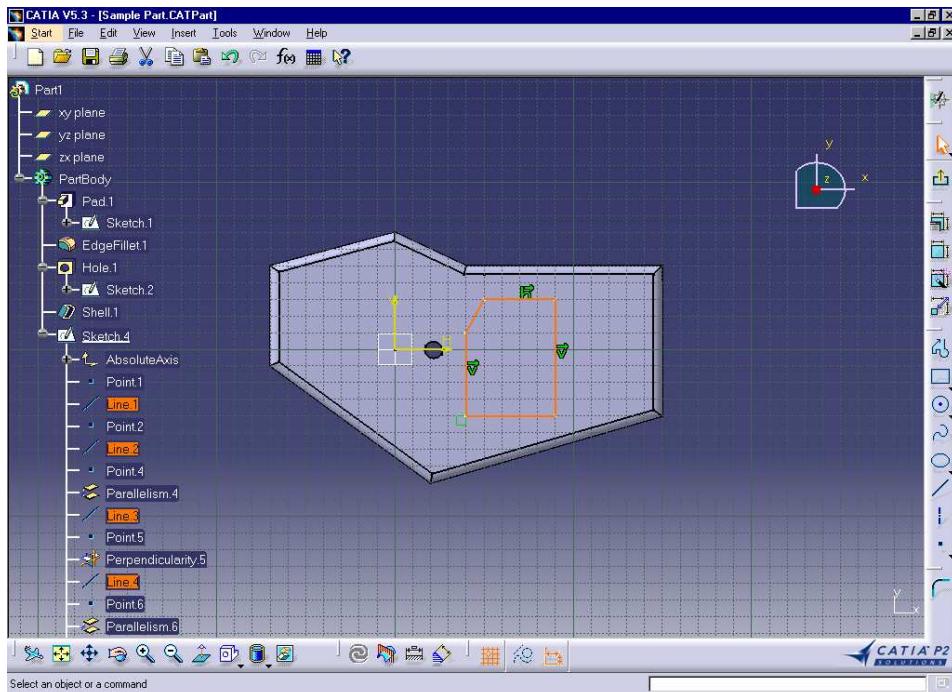
- To have a pointer on this method, you have to use the static method:
 - ◆ static CATFrmEditor * GetCurrentEditor()
- Thanks to this class, you can handle:
 - ◆ Compass
 - perform manipulations on CATIA objects (manipulate viewpoints, move and rotate non-constrained objects, ...)
 - ◆ CSO (Current Set of Objects)
 - contains highlighted objects
 - ◆ HSO (Highlighted Set of Objects)
 - contains objects that the command highlights (each object in the CSO is also in the HSO)
 - ◆ PSO (Pre-selected Set of Objects)
 - contains objects that are handled by a manipulator set by the current command, and that are pre-activated and moved
 - ◆ ISO (Interactive Set of Objects)
 - contains objects which are not part of the document, but which are displayed to enable their document object handling, such as manipulator handles.
- You can also retrieve:
 - ◆ Current document: CATFrmEditor::GetCurrentDocument()
 - ◆ Active object: CATFrmEditor::GetUIActiveObject()

Note: you can also handle the document thanks to **CATILinkableObject** interface from almost any objects

Path To Objects class: CATPathElement

Student Notes:

- ◆ Access an object is done thru its ancestors.
- ◆ Ex: when selecting a line defined in a sketch, the complete path to access the line is retrieved through a **CATPathElement= \Part1\PartBody\Sketch4\Line**

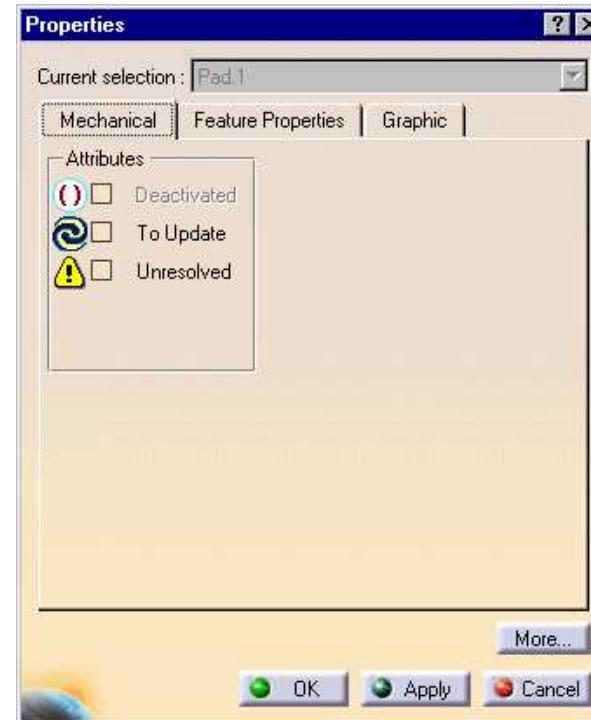


Student Notes:

Object Properties

- Object Properties edition can be triggered thru :
 - ◆ Selecting it, then 'Edit' menu bar
 - ◆ Selecting it, then pressing Alt+Enter
 - ◆ contextual menu item

- To create your own tab page in in the object property panel, see the use case in CAA V5 Encyclopedia
 - ◆ User Interface / Wintop Frame / Creating a Property Page for Object Properties

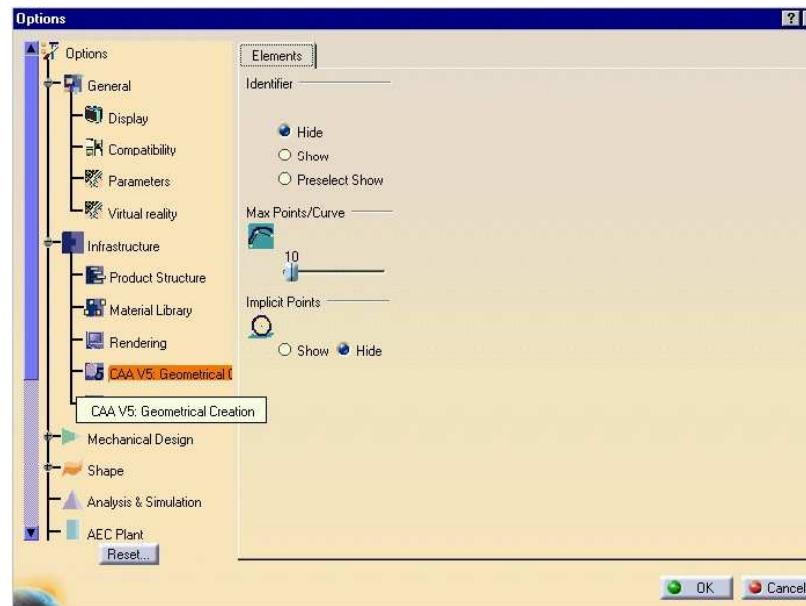


Student Notes:

Application Properties

- Application Properties are edited thru the Tools+Options command:

- All global parameters are specified here, some are specific to an application, most are shared among all of them.
- Their values are kept in «settings» repositories sometimes both in administrator reference and in user preferences
 - C:\Winnt\Profiles\User\Application Data\Dassault Systemes\CATSettings



- To create your own tab page in the application properties panel, see the use case in CAA V5 Encyclopedia
 - User Interface / Wintop Frame / Creating a Property Page for Application Properties

Student Notes:

Workshop, Workbench, Addin

- A **workshop** (as a workbench) defines a list of commands accessible through menus or toolbars.
 - ◆ A workshop defines the common menus and toolbars associated to a given document type (exception : sketch).
 - ◆ A workbench provides specialized commands.
 - Part Design for solid modeling
- Several workbenches may be associated to a given workshop
 - ◆ For the CATPart document, several workbenches are available:
 - Part Design, Wireframe and Surface Design, Free Style, Generative Shape Design
- An **Addin** represents a toolbar or menu in an existing workbench

Student Notes:

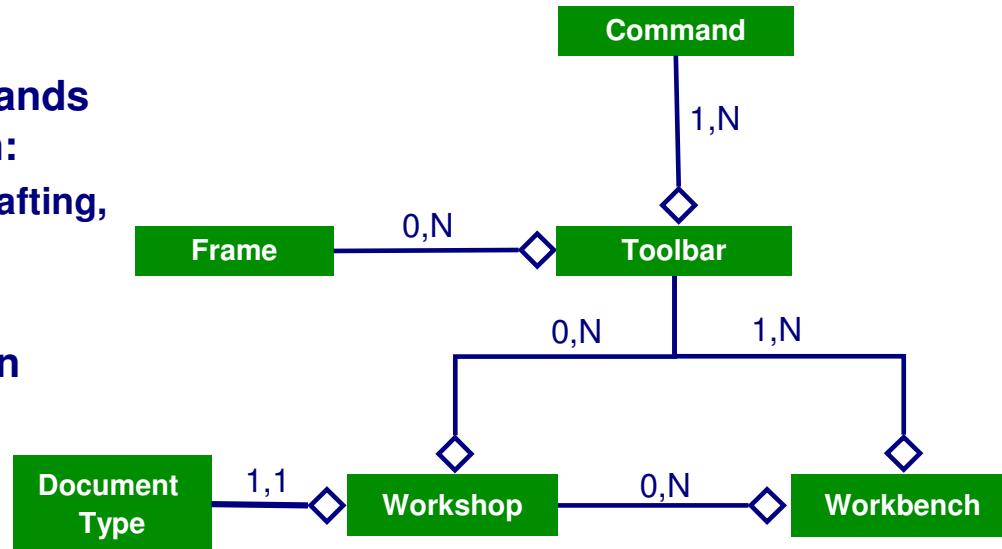
A Model for Command Access

- A command encapsulates a basic user action:
 - ◆ Creation, Edition, Analysis, Deletion

- A workbench groups commands dedicated to a given domain:
 - ◆ Part Design, Generative Drafting, Assembly Design, ...

- A workshop groups common commands to a set of workbenches for a given document type

- The frame provides the generic commands for all the workshops



Student Notes:

Command Header

- Command header hold necessary information to load the command.
- You can create your own customized command Header by creating its class, callbacks, etc... (see Creating Customized Command Header in CAA doc).

MyWorkbench.cpp

```
#include "CATCommandHeader.h"
MacDeclareHeader( CATMyCmdHeader )  
  
void MyWorkbench::CreateCommands()
{
    new CATMyCmdHeader( "CommandName" ,
                        "SharedLibraryDefiningTheCommand" ,
                        "CommandClassName" ,
                        (void *) DataToBePassedToTheCommand );
    ...
}
```

Macro declaring and defining a header class

Command Header instantiation

[Student Notes:](#)

Application Integration

■ Three ways to integrate your commands in the CATIA V5 frame:

◆ Define an Addin in a specific context:

- Add some toolbars in an existing workbench
- Create new Menus and Sub-Menus
- Insert buttons or menus in existing components

◆ Define a contextual menu :

- triggered from displayed objects or from the background only
- Creating new one(through Dialog command) or editing one through extension

◆ Define a new workbench linked to a specific workshop

- Define some toolbars
- Add some menus in the Main Menu Bar of CATIA

Student Notes:

Addin

- Define an object that implements the Addin interface of an existing workbench (Toolbar and Menu)

Example: CATICATSAMWorkshopAddin interface for the CATAnalysis document

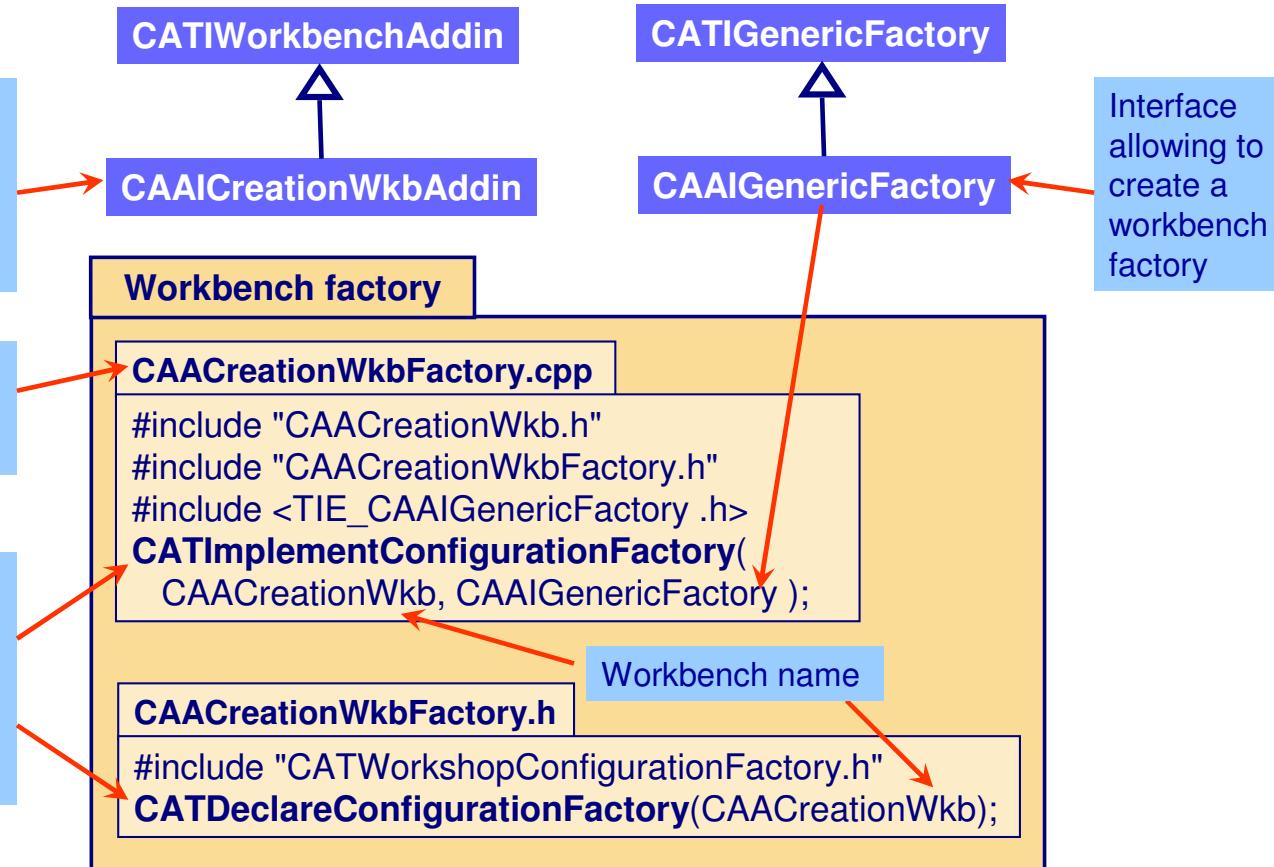
- Two methods to implement:
 - void AddinObject::CreateCommands()
 - CATCmdContainer * AddinObject::CreateToolbars()



Student Notes:

Workbench (1/4)

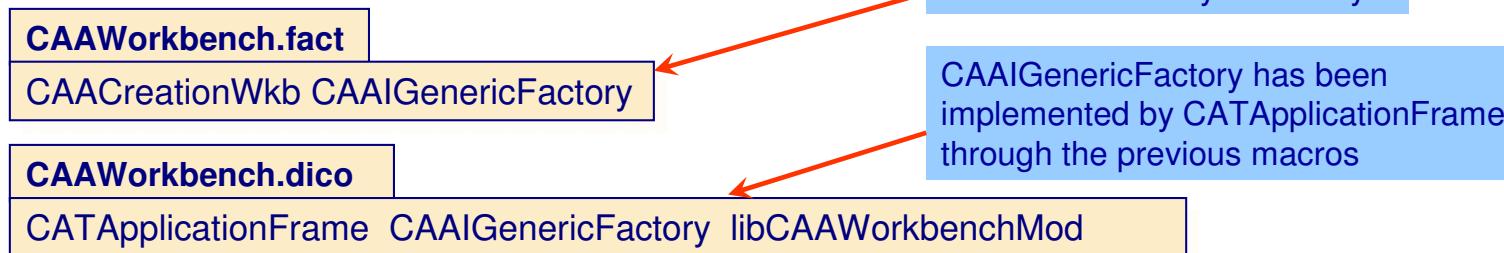
- First step: Workbench factory creation



Student Notes:

Workbench (2/4)

- Second step : Update the dictionary



- Third step: Implement the workbench interface methods:

- ◆ void WorkbenchObject::CreateCommands()
- ◆ CATCmdWorkbench * WorkbenchObject ::CreateWorkbench()
- ◆ CATClassId CAACreationWkb::GetAddinInterface()



- Fourth step: Add the workbench in a specific package (like Infrastructure) if you put the following file resource in the resources/msgcatalog folder:

```

CAACreationWkb.CATRsc
CAACreationWkb.Category = "Infrastructure";

```

[Student Notes:](#)

Workbench (3/4)

Last Step: Implementation of the CreateWorkbench() method

MyWorkbench.cpp

```
...
CATCmdWorkbench * MyWorkbench::CreateWorkbench()
{
    NewAccess(CATCmdWorkbench,pMyWorkbench,NewWorkbench);  
  

    NewAccess(CATCmdContainer,pMyContainer,NewToolBar);  
  

    NewAccess(CATCmdStarter,pStarter1,NewButton1);
    SetAccessCommand(pStarter1,"MyCommand1");
    SetAccessChild(pMyContainer,pStarter1);  
  

    NewAccess(CATCmdStarter,pStarter2,NewButton2);
    SetAccessCommand(pStarter2,"MyCommand2");
    SetAccessNext(pStarter1,pStarter2);  
  

    AddToolbarView(pMyContainer,1,Right);
    SetAccessChild(pMyWorkbench,pMyContainer);
...  

```

Define the workbench

Define a starter that will be associated to a command header and then arrange the command order

Define the container as a toolbar, position it by default on the right and include it in the workbench

[Student Notes:](#)

Workbench (4/4)

- Additional step: Complete an existing Menu in the main menu bar of CATIA

```
Workbench.cpp
...
CATCmdWorkbench * MyWorkbench::CreateWorkbench()
{
...
    NewAccess(CATCmdContainer,pINSERT,INSERT);
    NewAccess(CATCmdContainer,pMyMenu, MyMenu);
    SetAccessChild(pMyMenu,pINSERT); ← Retrieve the Insert Menu
    ...
    NewAccess(CATCmdStarter,pMyCommand1,MyCommand1);
    SetAccessChild(pINSERT, pMyCommand1);
    SetAccessCommand(pMyCommand1,"MyCommand");
    ...
    SetWorkbenchMenu(pMyWorkbench, pMyMenu); ← Define another container to
                                                be aggregated in the Insert
                                                menu
    ...
    return pMyWorkbench;
}
```

Retrieve the Insert Menu

Define another container to
be aggregated in the Insert
menu

Define the container as a
menu in the workbench

Exercise Presentation → Screw : Second part

And now practice on exercise, to learn about:

◆ Screw Creation in interactive mode :

- Create your toolbar
- Create an One Shot command and Implement the activate method
- Retrieve the factories from the editor
- Create geometry using DS feature (Pad and Shaft)
- Use your interactive command inside CATIA V5

Student Notes:

Student Notes:

CAA V5 Dialog

In this course you will learn the CATIA V5 infrastructure to build Dialog Boxes

- Framework objectives
- Building graphic user interfaces
- Retrieving user inputs
- The Dialog builder

[Student Notes:](#)

Framework objectives

- ❖ **Programmer productivity**
 - ◆ High level objects and widgets
 - ◆ Promotion of reusable components
- ❖ **Portability**
 - ◆ Abstract objects for:
 - UNIX
 - MS WINDOWS
- ❖ **Standard compliance**
 - ◆ Built on top of high level native software
 - OSF/MOTIF
 - MFC
- ❖ **Versatility**
 - ◆ Dialog applications can be run in:
 - CATIA V4 (DS only) - CATIA V5
 - Stand alone

[Student Notes:](#)

Provided Objects

Containers

- Used to group, structure, and present component objects into dialog windows.
- Different types:
 - Window
 - Bar
 - Menu
 - Box

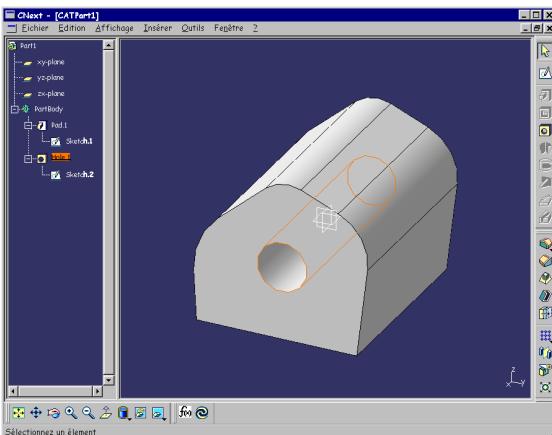
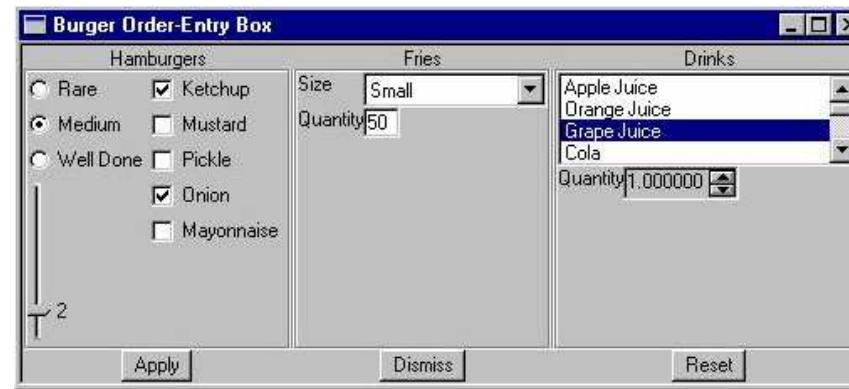
Components

- The building blocks of the Graphic User Interface.
- Different types:
 - Control
 - Menu Item

Student Notes:

Container: Windows

- A window is an independent resizable container in which other Dialog objects are created.

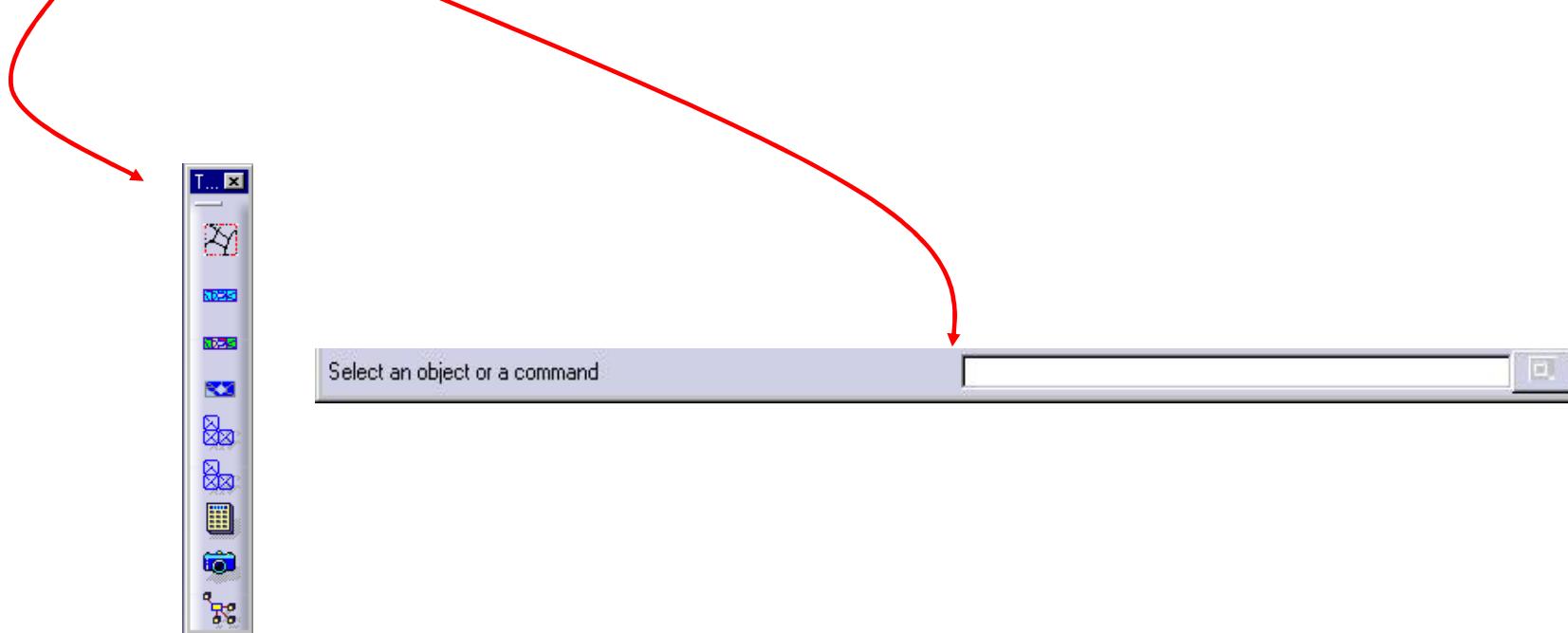
CATDIgDocument**CATDIgDialog****CATDIgNotify****CATDIgFile**

Student Notes:

Container: Bars

- A bar is a standard permanent zone for task starter or information fields.

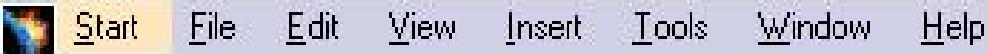
Class	Usage
<i>CATDlgtToolBar</i>	<i>Zone with direct access to application tasks</i>
<i>CATDlgtStatusBar</i>	<i>Zone for transient or permanent information</i>



[Student Notes:](#)

Container: Menus

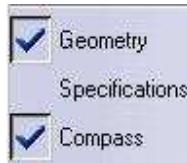
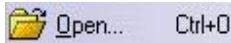
- A menu is a pop-up container for permanent task starter.

<i>Class</i>	<i>Usage</i>
CATDlgBarMenu	<i>Bar for main application menus</i>
	
CATDlgSubMenu	<i>Pop-up menu</i>
	
CATDlgContextualMenu	<i>Contextual menu</i>

Student Notes:

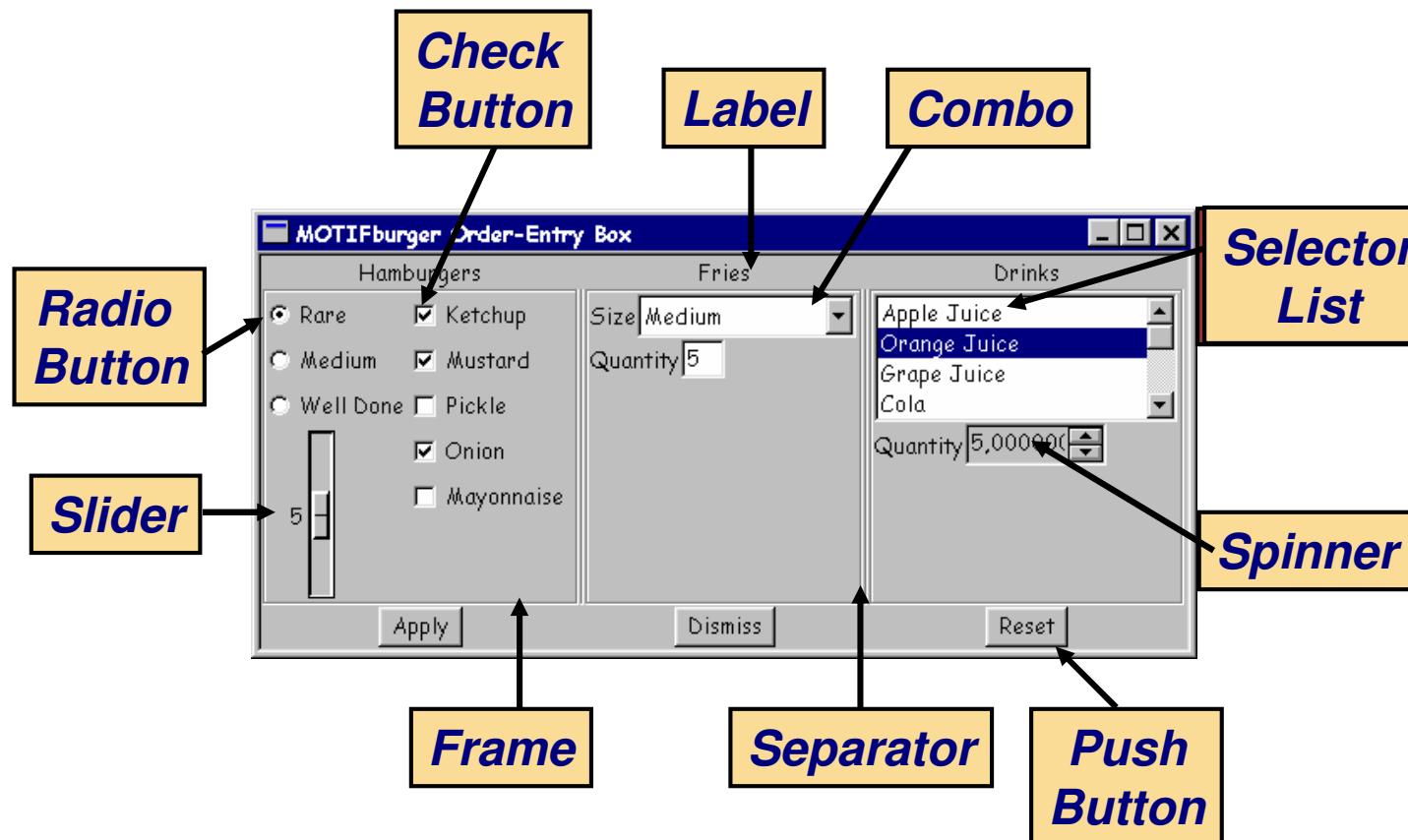
Component: Menu Items

- A menu item is a pop-up control for permanent task starter.

<i>Class</i>	<i>Usage</i>	
<i>CATDlgCheckItem</i>	<i>On/Off button in a menu</i>	
<i>CATDlgPushItem</i>	<i>Command button in a menu</i>	 
<i>CATDlgRadioItem</i>	<i>Radio button in a menu</i>	
<i>CATDlgSeparatorItem</i>	<i>Separation line in a menu</i>	

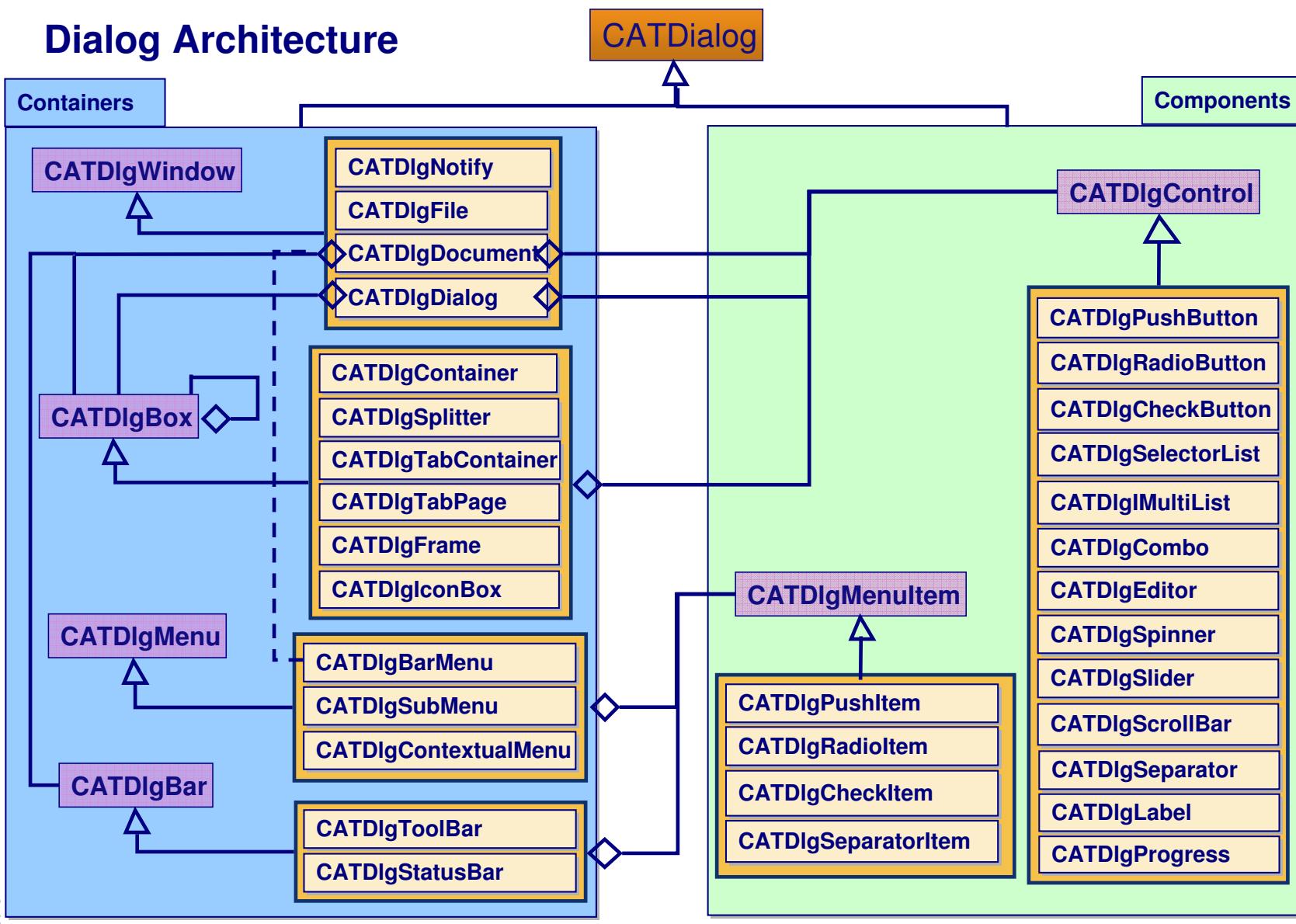
Component: Control

Student Notes:



[Student Notes:](#)

Dialog Architecture



[Student Notes:](#)

Dialog Design Steps (1/2)

- First, determine the type of your dialog
 - ◆ It is a dialog box or a window that contains several representations of a document.
 - Create a class that derives from CATDlgDialog.
 - ◆ It is a message pop-up.
 - Instantiate the CATDlgNotify class
 - ◆ It is a file selection box.
 - Instantiate the CATDlgFile class.
 - ◆ It is an application main window.
 - Create a class that derives from CATDlgDocument.

- Add a menu bar, a status bar, etc...
 - ◆ these objects are provided by the Dialog framework or by the ApplicationFrame framework.

Student Notes:

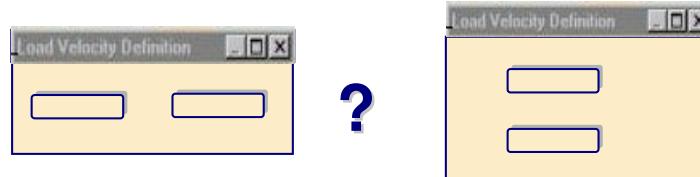
Dialog Design Steps (2/2)

- **Second, design the dialog appearance.**
 - ◆ Controls, frames, labels...
 - ◆ User interactions and actions.
 - ◆ Specifies the layout of the dialog.
- **Third, implement callback methods.**
 - ◆ A callback is called in response to a user interaction with a control.
 - ◆ You should provide one callback per user interaction.
- **Fourth, provide dialog resources.**
 - ◆ Resources are text and graphics displayed in the dialog.
 - ◆ Using resources facilitate translation.

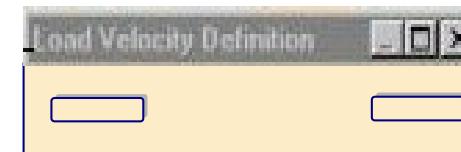
Student Notes:

Layout Management (1/2)

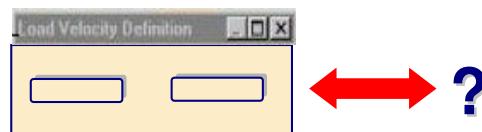
- The layout defines the way the objects are organized inside a container.



- when the container is initially displayed



- when the container's window is resized



Student Notes:

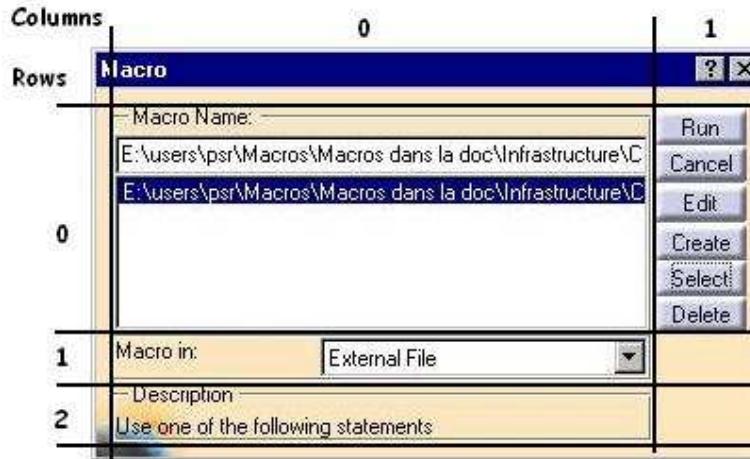
Layout Management (2/2)

- There are two kinds of layout management:
 - ◆ Grid layout management
 - Container is divided into cells, dialog objects are placed inside a cell,
 - easiest to use.
 - ◆ Attachment layout management
 - Dialog objects are dispatched in containers,
 - for more complex dialog.
- Dialogs generated by the Dialog Builder use the grid layout.

Student Notes:

Grid Layout Management (1/2)

- Objects are placed within a grid:



- To use the grid layout, you need to:
 - Create a container using the **CATDlGGridLayout** style


```
_pcontainer = new CATDlGFrame(this,"MyContainer",CATDlGGridLayout);
```

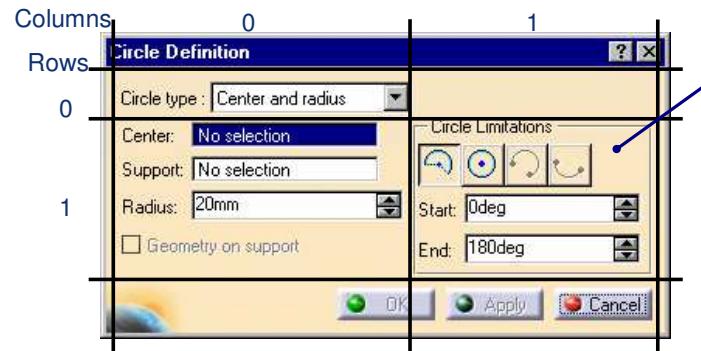
 - If the option **CATDlGGridLayout** was not here, the default layout will be attachment layout.
 - State where to place each dialog object inside the container
 - State how to attach each dialog object with respect to the cell sides
 - Enable rows and columns for resize.

Student Notes:

Grid Layout Management (2/2)

- To place a dialog object you have to set the grid constraint on the dialog object:

```
pCircleLimitationsFrame->SetGridConstraints(1,1,1,1,CATGRID_4SIDES);
```



CircleLimitationsFrame:

- placed at row 1 and column 1,
- 1 row span and 1 column span,
- attached to four sides.

- Use the SetGridColumnResizable and SetGridRowResizable methods to specify which columns and/or rows will be sensitive to the container resize
 - By default rows and column are non resizable.
 - Example : first row and second column are set to be resizable.

```
SetGridRowResizable(0,1);
SetGridColumnResizable(1,1);
```

Column index

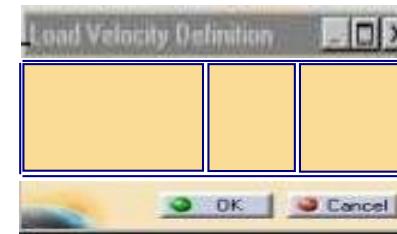
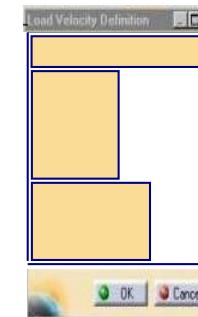
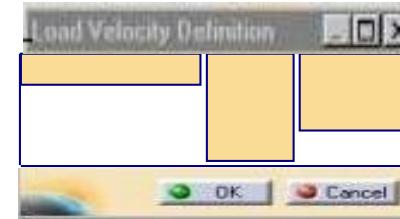
0 for non resizable
1 for resizable

Student Notes:

Attachment Layout Management (1/3)

- The Attachment Layout is based on tabulations to which objects are attached by one or several sides.
- By default, all the objects of a container are arranged side by side, horizontally, justified on top, in the order in which they are created.
- This default can be changed to Vertical by using the SetDefaultOrientation method:

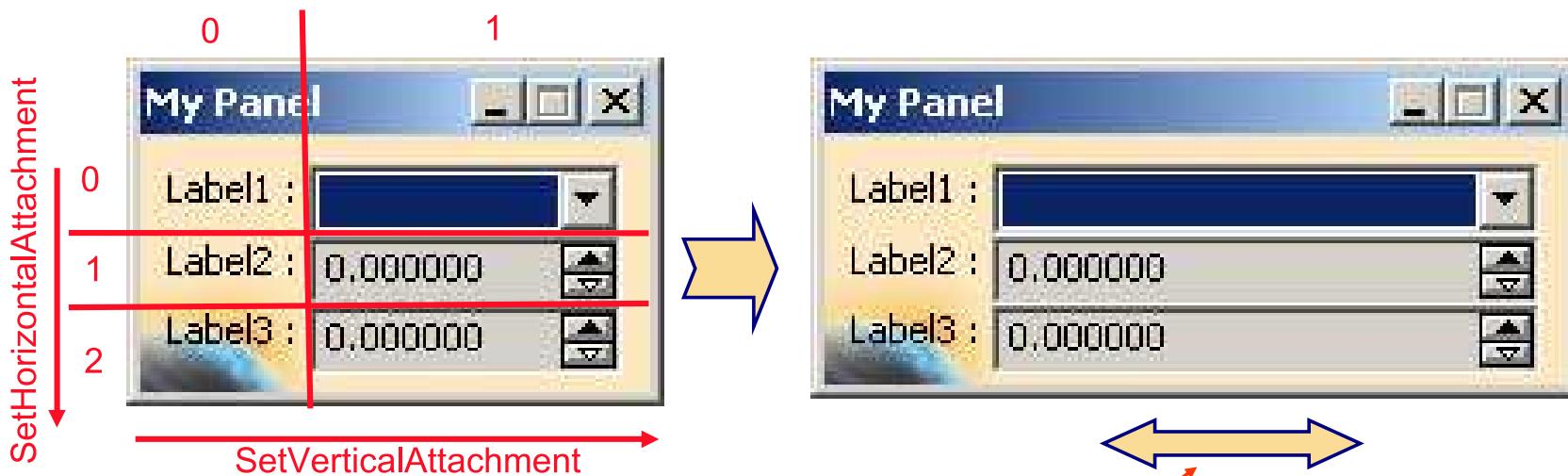
```
_pContainer ->SetDefaultOrientation(Vertical);
```



Student Notes:

Attachment Layout Management (2/3)

- Once the objects have been creating in the container, it is necessary to arrange them in the container. Note that arranging is not available for some containers such as CATDlgBar, for which the layout is imposed.
- Arranging can be done horizontally or vertically by calling **SetHorizontalAttachment** or **SetVerticalAttachment** methods



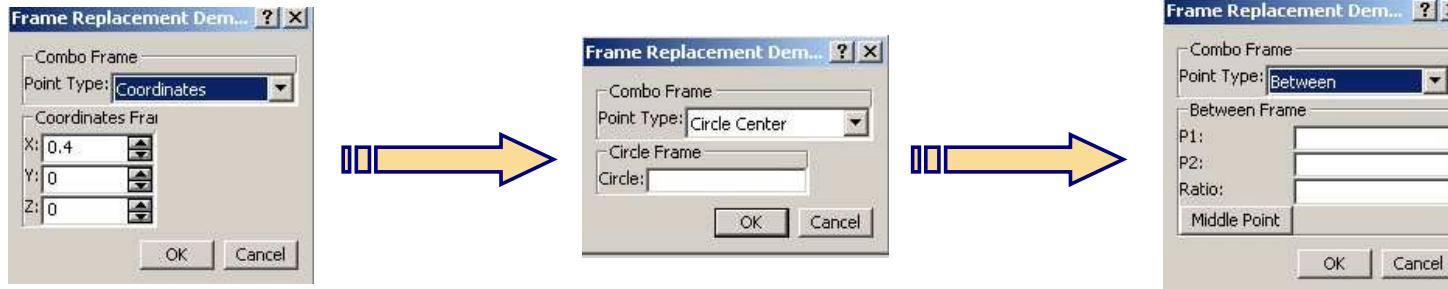
```
this->SetHorizontalAttachment(0, CATDlgTopOrLeft, pLabel1, pCombo1, NULL);
this-> SetHorizontalAttachment(1, CATDlgTopOrLeft, pLabel2, pSpinner1, NULL);
this-> SetHorizontalAttachment(2, CATDlgTopOrLeft, pLabel3, pSpinner2, NULL);
this-> SetVerticalAttachment(0, CATDlgTopOrLeft, pLabel1, pLabel2, pLabel3, NULL);
this-> SetVerticalAttachment(1, CATDlgRightOrBottomRelative, pCombo1, pSpinner1, pSpinner2, NULL);
```

Student Notes:

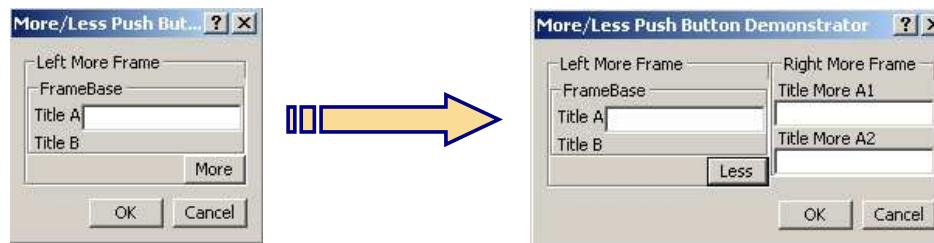
Attachment Layout Management (3/3)

- The Attachment Layout is the single way to have a dynamic management:

- Two or several objects can be design to be displayed at the same location, depending on the context.



- A dialog box can have a hidden part that can be displayed using a More push button.

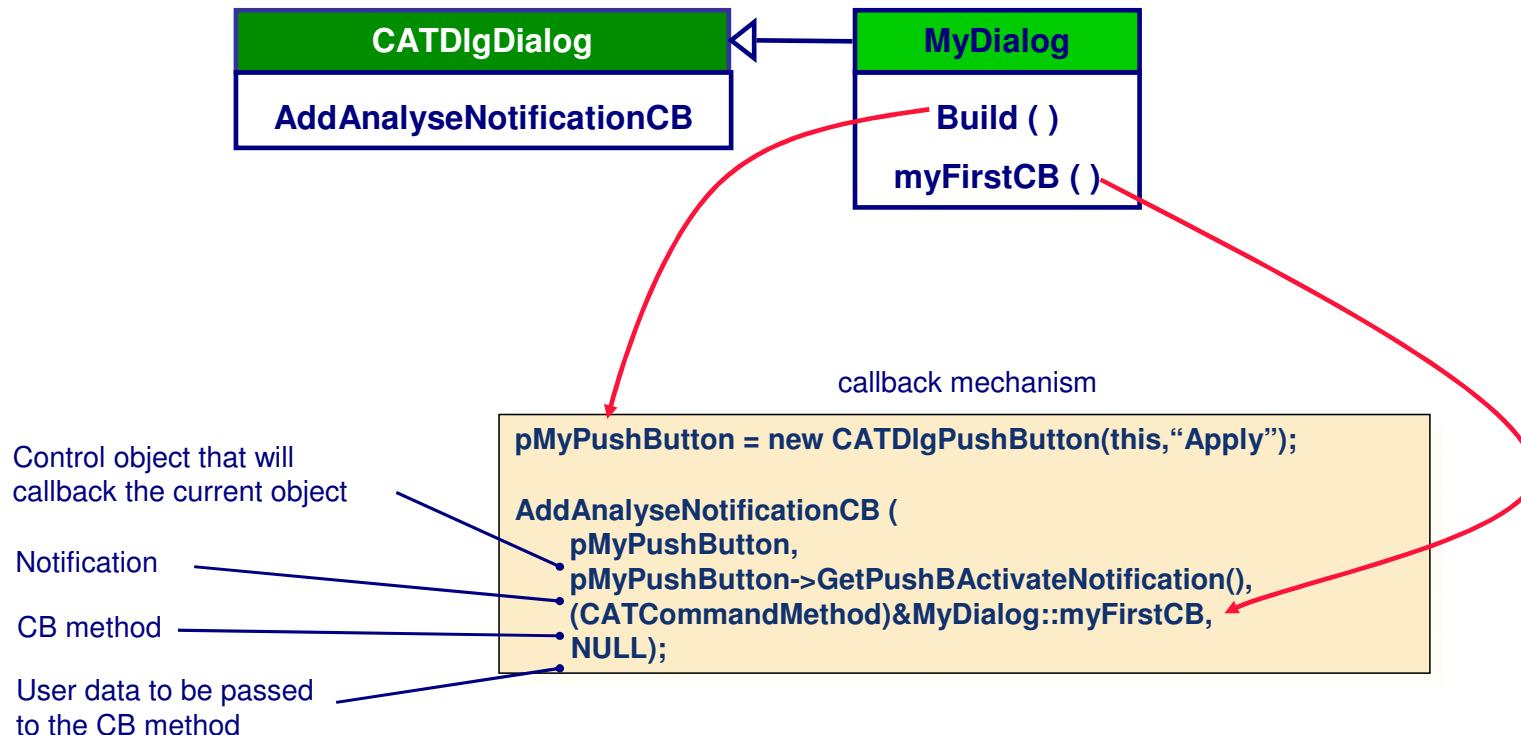


- For further information, you can see in the encyclopedia: « User Interface / Wintop Dialogs / Use Cases / Creating Dialog Boxes Automatically Resizable »

Student Notes:

Defining a Callback

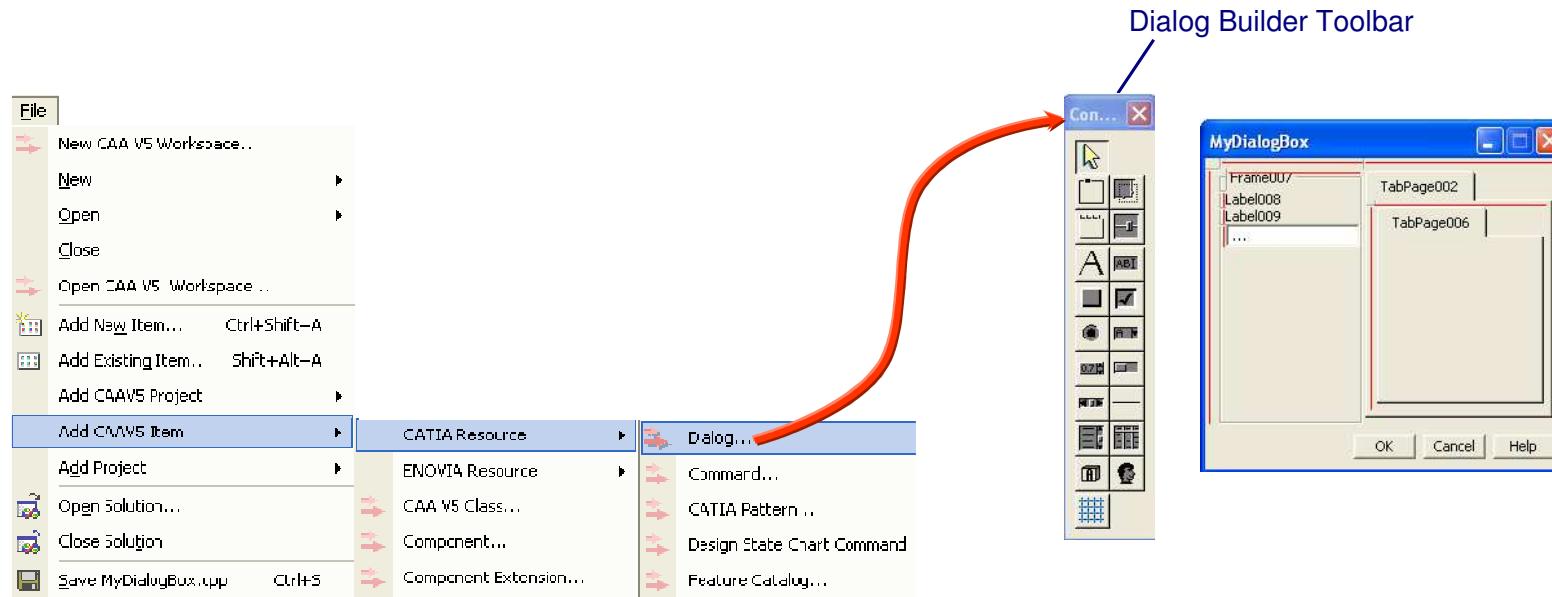
- Callbacks are set with the AddAnalyseNotificationCB method



Student Notes:

The Dialog Builder

- CATIA Dialog Builder makes panel creation and edition easier.
- Access through menu or by double-clicking on an existing *.CATDlG file.
- Generates C++ code.
- Possibility to define Callbacks.



[Student Notes:](#)

CAA V5 DialogEngine

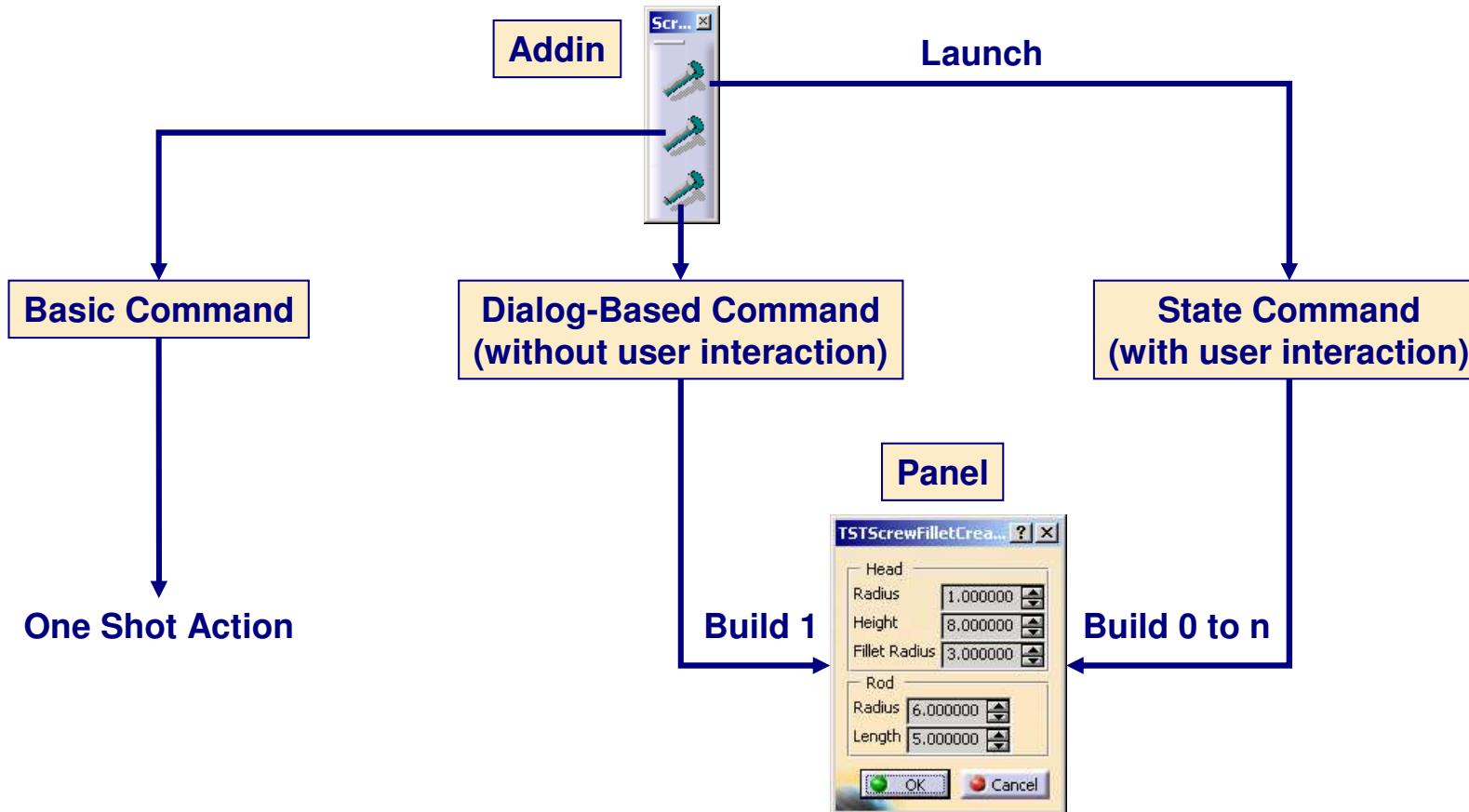
In this course you will learn the mechanisms necessary to describe and to manage the dialog of an interactive command, how to monitor the interactions at run time and manage Undo/Redo capabilities

- Dialog Engine Architecture
- Main notions
- How to define a new interactive command

[Student Notes:](#)

Dialog Engine Architecture

- Your new toolbar will launch a command who will be responsible for user interactions. If you need it, your command can manipulate a panel.

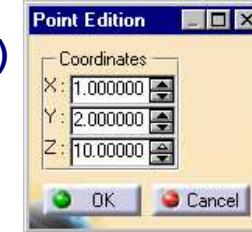


[Student Notes:](#)

Notions to describe a dialog

In CATIA, you have three main types of commands:

- If you don't need to select objects or to enter values, use a Basic Command.
 - ◆ (Example: Load a Feature catalog)
- If you don't need to select objects but you need to enter values, use a Dialog-Based Command.
 - ◆ (Example: Create a 3D Point by valuating X, Y and Z in a Panel)
- If you need to select objects, use a CATStateCommand (with a Panel or not)
 - ◆ only this kind of command allows you to manage the undo.



Student Notes:

Command with/without an argument

```
CATCreateClass( MyCommand )
```

...

This macro is mandatory to be able to instantiate a corresponding command header in a toolbar or a menu

- ❖ Sometimes it's interesting to pass an argument to a command, so the command can be reused for creation and also for edition

```
CATCreateClassArg( MyCommand,CATISample )
```

```
MyCommand::MyCommand (CATISample * iArg)
```

...

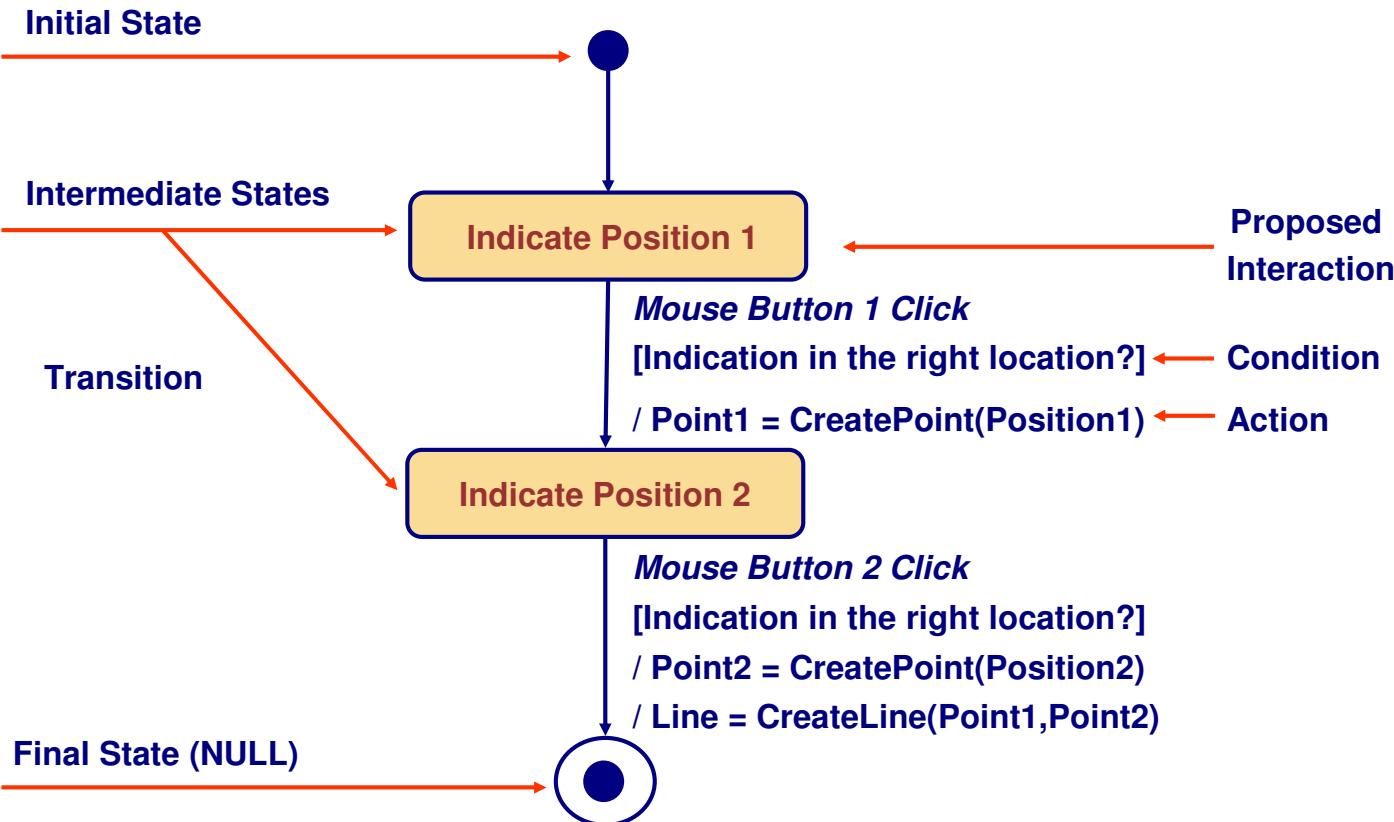
Student Notes:

Notions to describe a CATStateCommand

- A StateCommand will receive user interactions in CATIA ApplicationFrame.
- To manage this command, we use three main notions :
 - ◆ Agents, to get the user notifications and selected objects;
 - ◆ States, step in a dialog where the program is waiting for an input;
 - ◆ Transitions, to define when an action should be executed
- The type of the input is defined at each state.
- A Transition is defined between a source state and a target state.
- It is triggered by a Condition:
 - ◆ at least event-driven: an end user interaction
 - ◆ conditional, it validates the user input
- When a Transition is triggered, the target state becomes the active state.
- An Action can be executed during a transition.

Finite State Machine

Student Notes:



Student Notes:

The CATStateCommand Class

- Create a new class deriving from CATStateCommand
- Overload at least:
 - ◆ To describe your own state chart
 - BuildGraph()
 - ◆ To manage properly your command life cycle
 - Activate()
 - Desactivate()
 - Cancel()
 - ◆ Define some specific methods that will be used as conditions or actions
 - ◆ Store as data members the dialog agents used in the command

Student Notes:

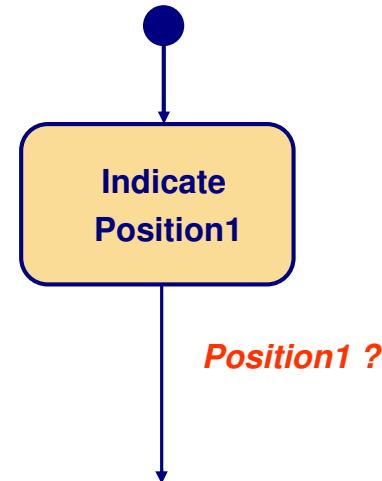
Define the State Charts

- Overload the BuildGraph method
 - ◆ Define the dialog agents you need
 - ◆ Create all the states and plug the dialog agents in the corresponding states
 - ◆ Define the transitions between states:
 - ◆ Source and target states
 - ◆ Condition
 - ◆ Action

Student Notes:

Agent (1/2) : Definition

- A Dialog Agent translates a user interaction into an user input
 - ◆ Ex: a CATIndicationAgent converts a left button mouse click in a 2D viewer as a 2D-coordinate input
- It hides the details of how a user interaction is converted as a user input.
- A dialog agent can be reused by recycling it after retrieving the input.
 - ◆ Don't forget to initialize its acquisition with **InitializeAcquisition()** method.



Student Notes:

Agent (2/2) : Main Types

- CATDialogAgent is the main class
 - This class can be used to define an agent linked to a panel object instantiated in the command.
 - This agent can be evaluated when a notification is sent.

```
_AgentPanelOK = new CATDialogAgent(" AgentPanelOK ");
_AgentPanelOK->AcceptOnNotify( _MyPanel, _MyPanel->GetDiaOKNotification());
```

The agent will be evaluated when the OK button of the panel will be pressed

- CATIndicationAgent retrieves the coordinates of a 2D Point when clicking in a viewer.
- CATPathElementAgent retrieves the path element of the object under the mouse when clicking

```
_myAgent = new CATPathElementAgent( "myAgent" );
_myAgent ->AddElementType (CATISample::ClassName());
_myAgent ->SetBehavior(CATDlgEngWithPSOHSO|CATDlgEngWithPrevaluation);
AddCSOClient( _ myAgent );
```

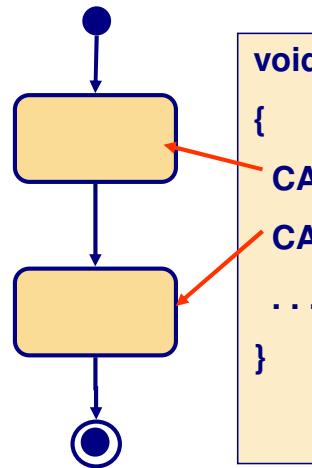
Limit the selection to objects that implement a given interface

Enable the Object / Action dialog style.
Look in the CSO (Current Set of Objects)

A valid object is highlighted when the mouse is located on it

States (1/2): Declaration

Student Notes:

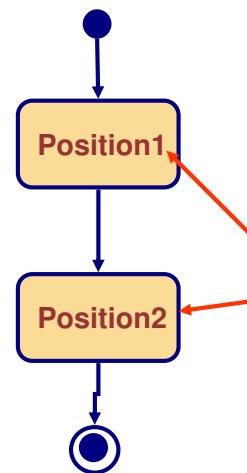


```
void MyCommand::BuildGraph()
{
    CATDialogState * State1 = GetInitialState("Start");
    CATDialogState * State2 = AddDialogState ("Second");
    ...
}
```

The initial state is already created.

States (2/2) : Plug the Dialog Agents to the states

Student Notes:



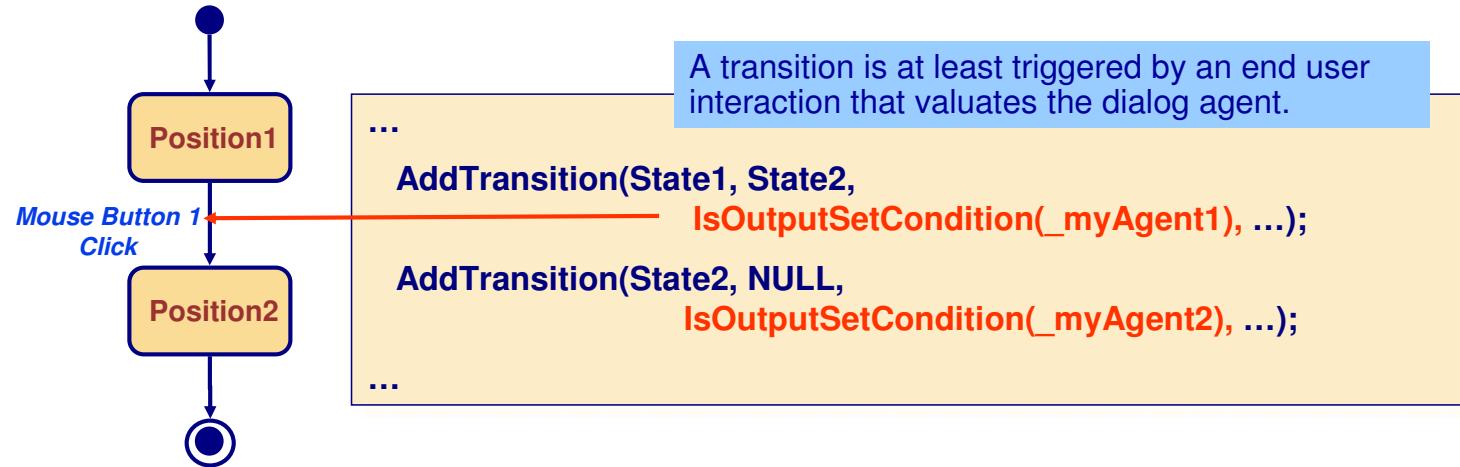
```
...
_myAgent1 = new CATPathElementAgent( "myAgent1" );
_myAgent2 = new CATPathElementAgent( "myAgent2" );

State1->AddDialogAgent(_myAgent1);
State2->AddDialogAgent(_myAgent2);
```

Several agent can be plugged
to the same state.

Student Notes:

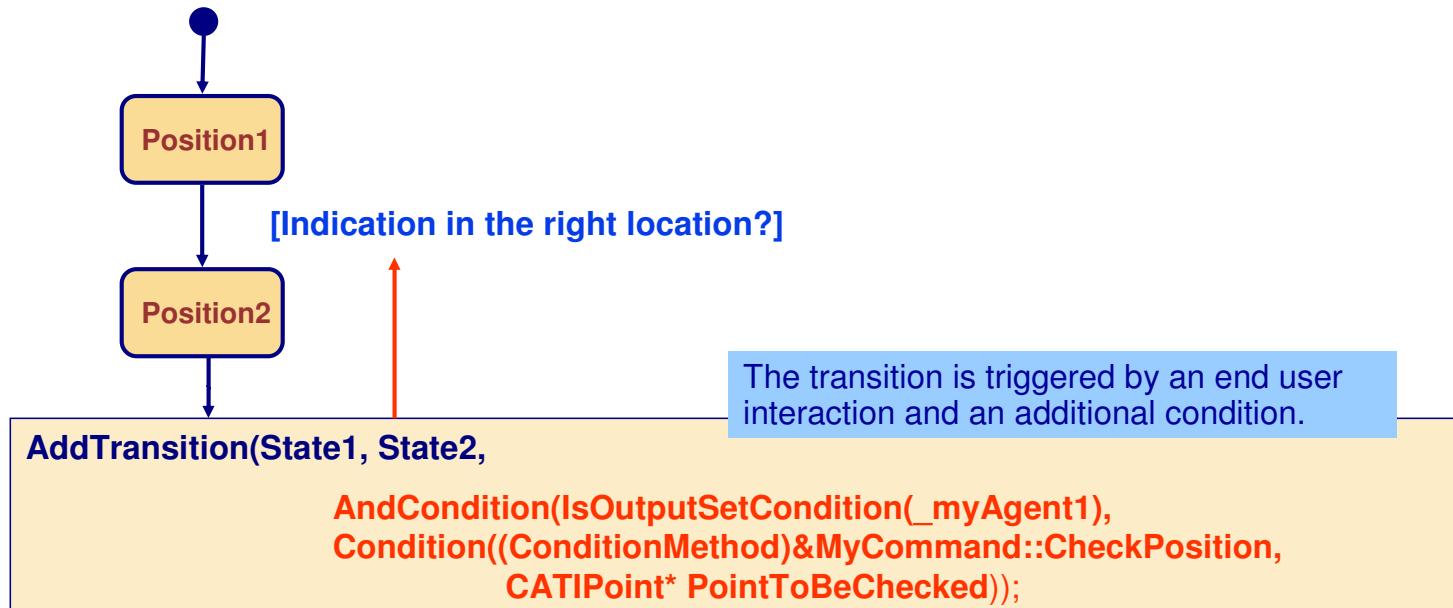
Transitions (1/3) : Agent condition



Student Notes:

Define the Transitions (2/3) : additional condition

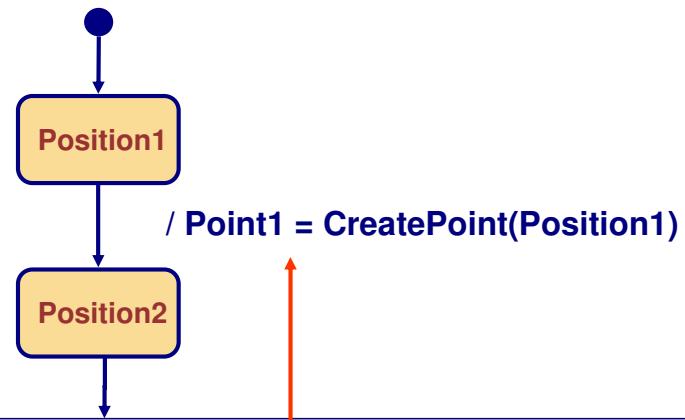
- The condition method signature is:
 - boolean ConditionMethod (void* iUsefulData)



Student Notes:

Define the Transitions (3/3) : actions to be executed

- The action method signature is:
 - boolean ActionMethod (void* iUsefulData)



/ Point1 = CreatePoint(Position1)

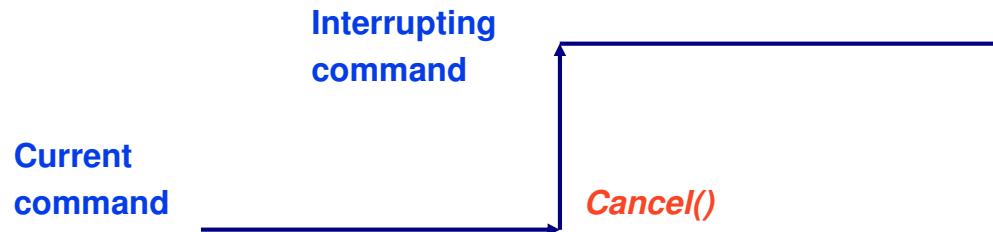
```
AddTransition(State1, State2, AndCondition(IsOutputSetCondition(_myAgent1),  
Condition((ConditionMethod)&MyCommand::CheckPosition,  
CATIPoint* PointToBeChecked)),  
Action((ActionMethod) &MyCommand::CreatePoint,  
(ActionMethod) &MyCommand::UndoCreatePoint,  
(ActionMethod) &MyCommand::RedoCreatePoint,...) );
```

Student Notes:

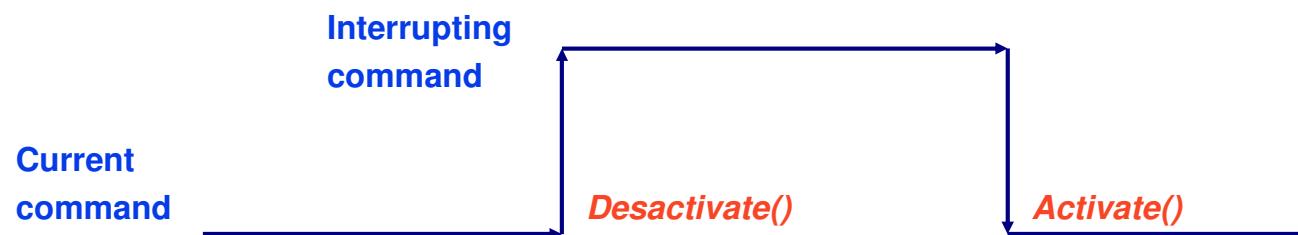
Interruption by another command: Exclusive or Shared

- Only for CATStateCommand: the second command can be launched interactively or can be called by a simple new() in the first command.

- The interrupting command is exclusive



- The interrupting command is shared



Student Notes:

Launching a sub-command (or macro-agent)

- The **CATStateCommand** class derives from the **CATDialogAgent** class, then it can be used as a agent in another command.
 - ◆ The sub-command must be created with the **CATCommandModeUndefined** mode.
 - ◆ It must be the unique agent in a state.
 - ◆ It must have the **CATDlgEngWithUndo** behavior.
 - ◆ If the sub-command ends normally, the transition is executed.
 - ◆ If the sub-command is cancelled, the main command returns to the previous state.
 - ◆ Its lifecycle will be linked to the main command as a standard **CATDialogAgent**.

```
SecondCmd *_pSecondCmd = new SecondCmd();  
  
State1->AddDialogAgent(_pSecondCmd);  
  
AddTransition(State1, State2, IsOutputSetCondition (_pSecondCmd),  
             Action ((ActionMethod) &MyCmd::ActionOne));
```

Student Notes:

Additional Information: Propose Multi Acquisition (1/2)

- To enable multi-selection for a path element dialog agent, set its behavior to
 - CATDlEngMultiAcquisitionSelModes (selection by trap)
 - CATDlEngMultiAcquisitionCtrl (selection by trap and by Ctrl Key)
 - CATDlEngMultiAcquisitionUserCtrl (allow you to switch between the two previous modes)

```
...
_pAgentMultiSel = new CATPathElementAgent( "Sample" );
_pAgentMultiSel ->AddElementType(CATISample::ClassName());
_pAgentMultiSel ->SetBehavior(CATDlEngWithPSOHSO |
                                CATDlEngWithPrevaluation |
                                CATDlEngMultiAcquisitionCtrl);
_pAgentMultiSel->AddElementType(CATISample::ClassName());
AddCSOClient(_pAgentMultiSel);
...

```



Student Notes:

Additional Information: Propose Multi Acquisition (2/2)

- To Retrieve your set of selected objects from your path element agent you must use the CATSO class.

```
...
CATSO* pSOonSample = _pAgentSample->GetListOfValues();
if (NULL != pSOonSample)
{
    int NbOfSample = pSOonSample->GetSize();
    for (int i = 0 ; i < NbOfSample ; i++)
    {
        CATPathElement* pPEonSample = (CATPathElement*)(*pSOonSample)[i];
        CATBaseUnknown* piBUonSample = pPEonSample->FindElement(IID_CATISample);
        ...
    }
}
...
```

Method to retrieve the list of selected objects



Student Notes:

Additional Information: Rubber Banding

Define a self-transition

Use a Dialog Agent with specific behaviors

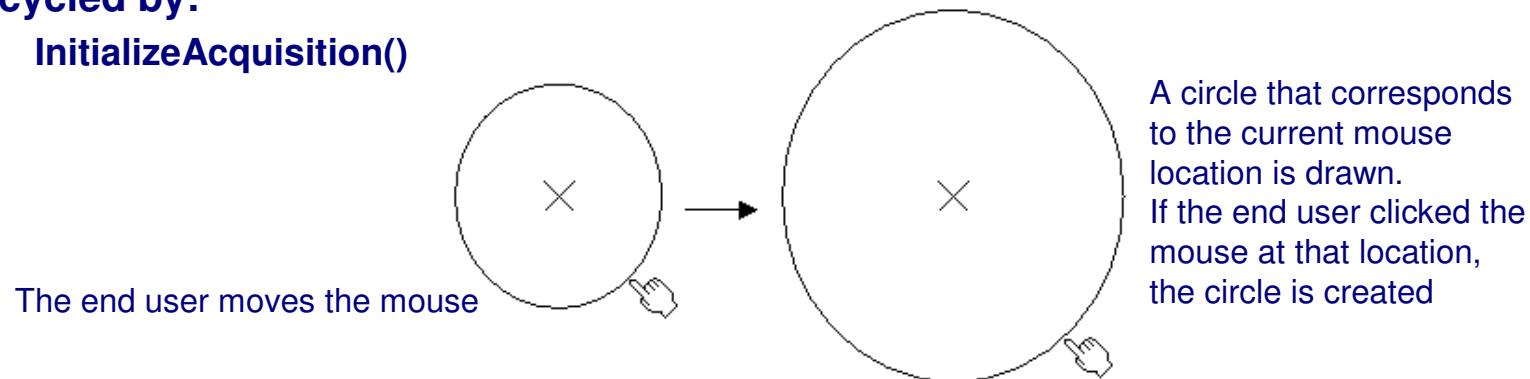
- CATDlgEngWithPrevaluation: the agent can be valued by the object that is under the mouse without selecting it.
- CATDlgEngAcceptOnPrevaluate: The transition is triggered when the agent is valued without selecting.

The condition to trigger the transition is:

IsLastModifiedAgentCondition(_myAgent)

After performing the corresponding action, the dialog agent needs to be recycled by:

InitializeAcquisition()



Student Notes:

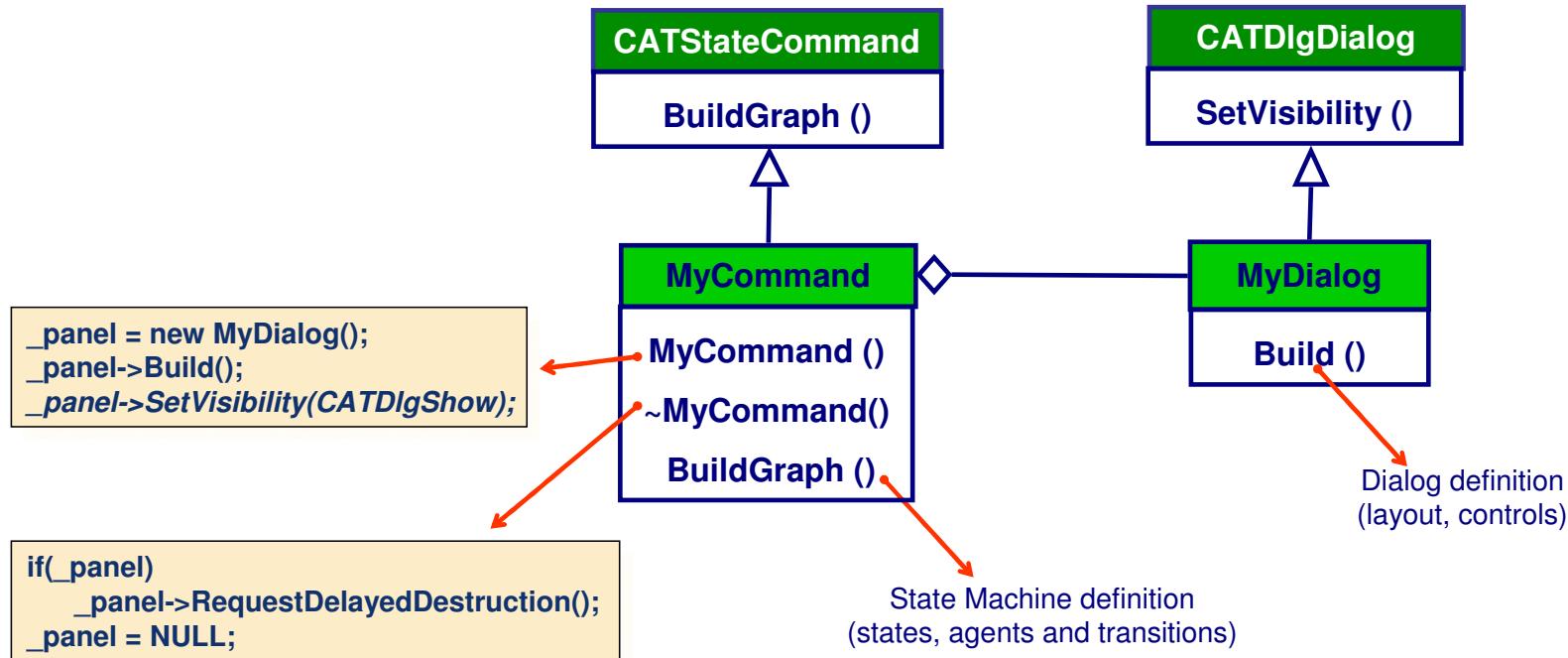
Additional Information: Managing Undo/Redo

- Two levels of Undo/Redo
 - ◆ Input Undo/Redo within a command
 - ◆ Command Undo/Redo
- When an action is associated to a transition, an Undo method should be defined
- Within a command if the Undo command is activated, the previous transition is inverted and if there was an action, the corresponding Undo method is run.
- Then, when out of a command, global Undo methods are executed to cancel the results of the previous commands

Student Notes:

Dialog Engine Integration (1/2)

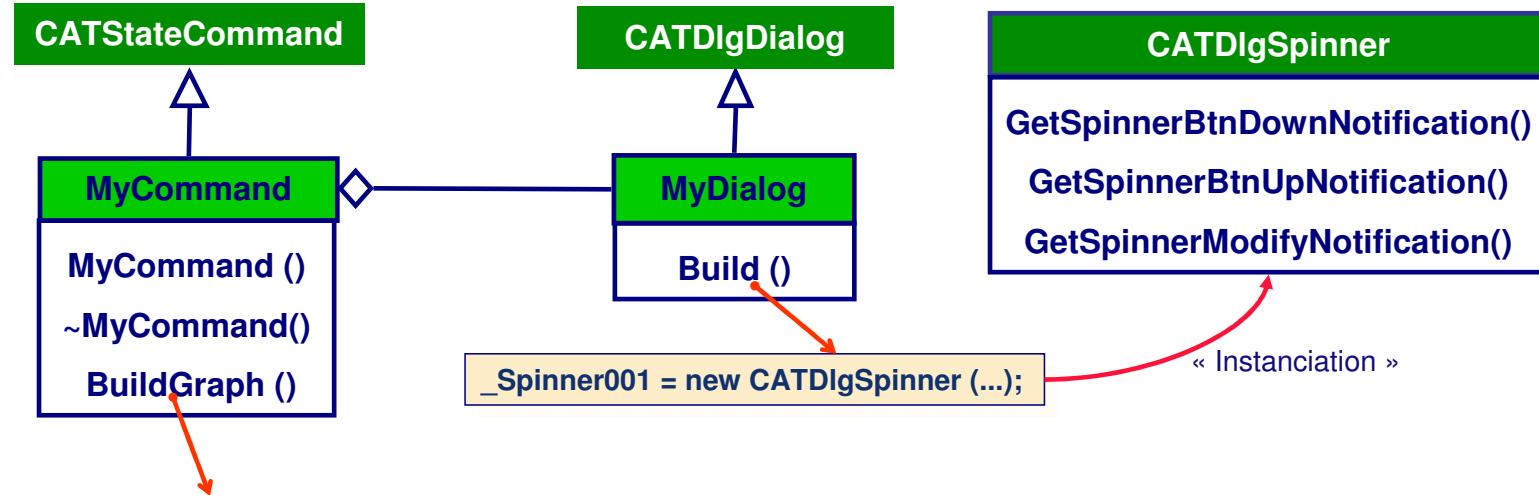
- Associate a dialog to your command



Student Notes:

Dialog Engine Integration (2/2)

- Use a DialogAgent instead of a callback when you want to perform a transition.



```

_spinnerAgent = new CATDialogAgent("Agent");
_spinnerAgent->AcceptOnNotify( _panel->_spinner001, _panel->_spinner001->GetSpinnerModifyNotification());

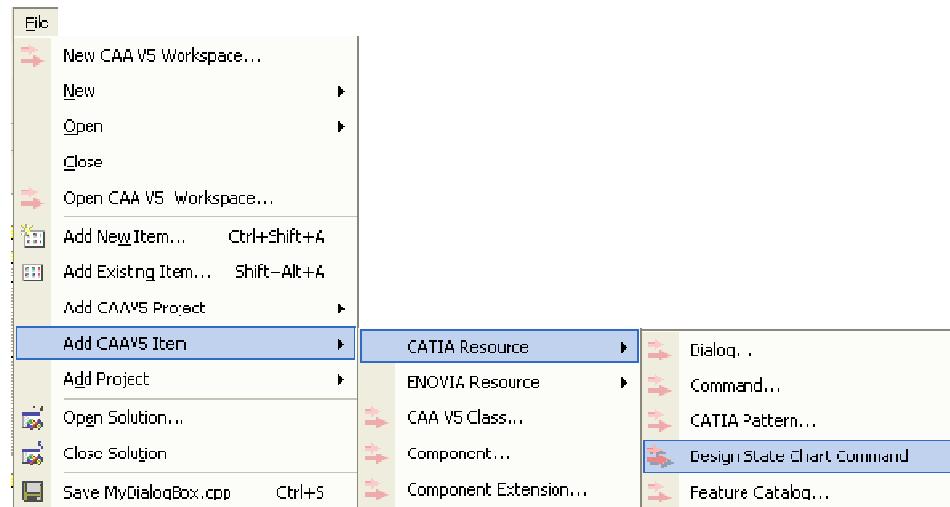
firstState->AddDialogAgent(_spinnerAgent);
...
AddTransition (firstState, secondState, IsOutputSetCondition(_spinnerAgent),
Action((ActionMethod)&MyCommand::MyActionMethod));
  
```

- If you do not need any transition for your dialog objects just use an AddAnalyseNotificationCB

Student Notes:

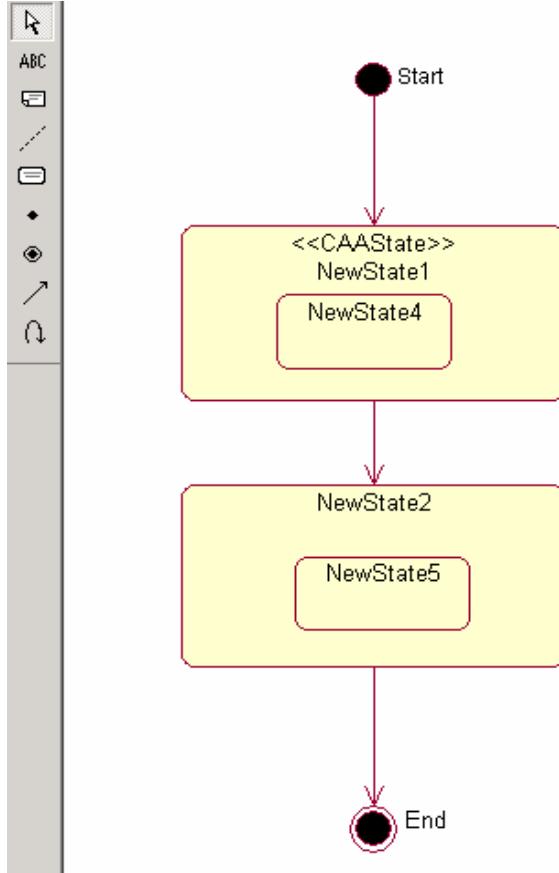
State Charts definition using DMC (1/3)

- It is also possible to use Data Model Customer (DMC) to create a state chart command.
- First of all, the state chart design:
 - ◆ In Visual C++, go to Insert\CATIA Resource\Design State Chart Command to launch Rational Rose



Student Notes:

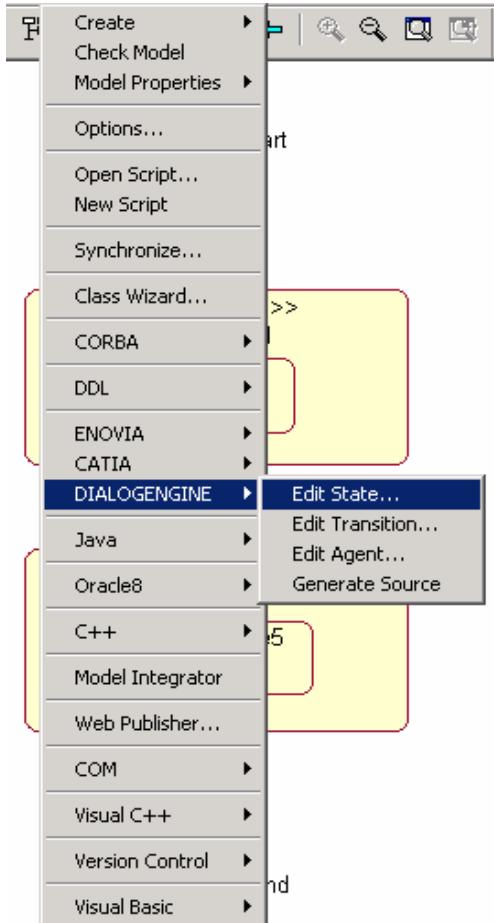
State Charts definition using DMC (2/3)



- ❖ Create the state chart components:
 - ❖ States (Example: NewState2)
 - ❖ Agents that are defined here as sub-states (example: NewState4)
 - ❖ Transitions
- ❖ By default the initial state (Start) and the first states (NewState1) are automatically created.
- ❖ A final state(End) is mandatory to go out the command.

Student Notes:

State Charts definition using DMC (3/3)



- ❖ Then define these components (output conditions for the state, agent type or action during a transition)
 - ❖ Tools\ DIALOGENGINE\ Edit....

- ❖ Call Tools\ DIALOGENGINE\ Generate Source command to generate automatically the C++ code associated with your state chart

Student Notes:

Dialog / Dialog Engine / Tool bar / Workbench

- All this objects are referenced by an unique identifier (his name). To avoid name collisions inside and outside the CAA environment, you must follow the naming convention (C++ class name, identifier, late type, ...).

Guides



Technical Articles

CAA Authorized APIs	Moving Authorized APIs and A control
CAA V5 Authorized API Identification, Usage, Deprecation, and Stability	Understanding CAA Authorized APIs
CAA V5 Naming Rules	Standard names for CAA V5 bi applications, products, and cc
CAA V5 Component Rules	Standard patterns for CAA V5
CAA V5 C++ Coding Rules	Rules, hints and tips to write c
CAA V5 C++ 64-bit Operating System Support	Rules, hints, and tips to help y bit operating systems
CAA V5 C++ Error Processing Rules	Rules, hints and tips to manag
CAA V5 C++ Naming Rules	Standard names for public CA
CAA V5 C++ Interface and Class Documentation	Hints and tips to tag and write files
Rules	Standard NLS rules for CAA V5
CAA V5 NLS Rules	

Rules and Standards

Technical Article

Abstract

As CAA V5 introduces a number of different programming entities, naming rules and conventions have been developed in order to avoid name collisions inside and outside the CAA environment, and to make things clearer for its developers. This article describes these naming rules for the C++ entities.

- General Principles
- Global Rule
- Entities Associated with a Directory
- C++ Entities Associated with a File
- Language-independent Entities Associated with a File
- User Interface Entities
- Other Entities
- Naming Variables Within C++ Code
 - General Rules
 - Hungarian Notation
 - Naming Interface Pointers
- References

CAA V5 C++ Naming Rules

Standard names for public CAA V5 C++ entities

Student Notes:

CAAV5 Resources

You will learn how to define your resources files (text, icon, ...).

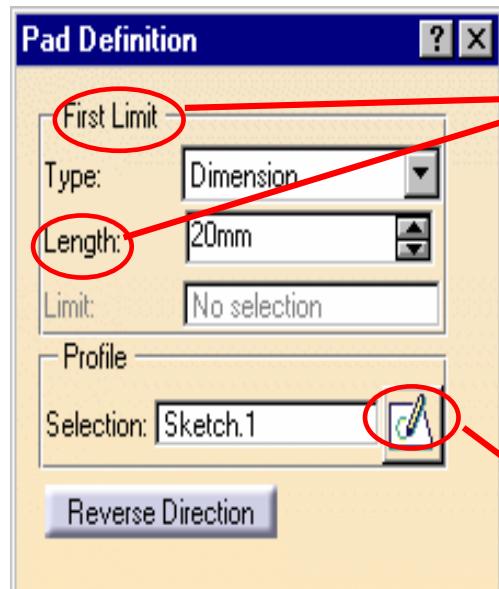
- Resources Overview
- Assigning Resources
- Assigning Predefined Resources
- CATDialog Predefined Resources

[Student Notes:](#)

Resources OverView

What are resources?

- ◆ Resources could be text or icons displayed by the application.
- ◆ Text resources are compatible with National Language Support
- ◆ Resources can be changed without recompilation



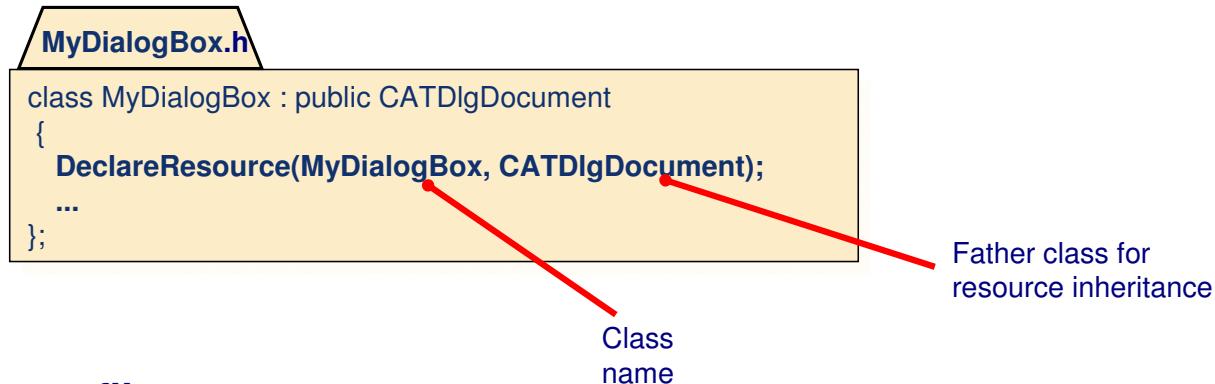
Text resources are stored in a CATNls file

Path to Icon resource is stored in a CATRsc file

Student Notes:

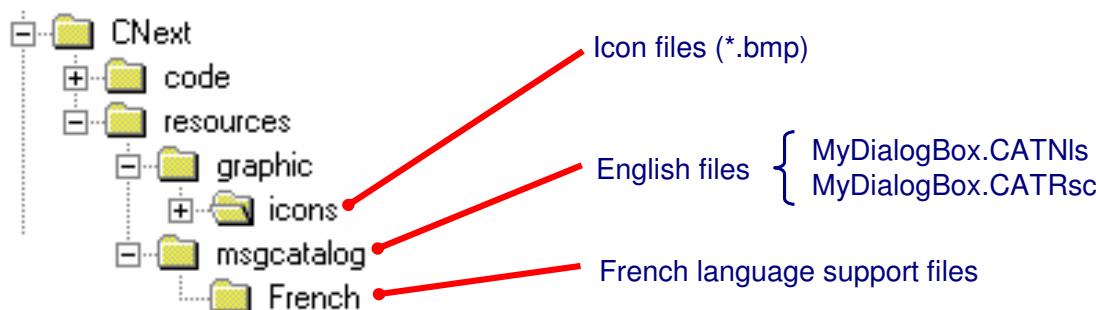
Assigning Resources: Declaration and Creation

Declare resources in your object



Create resource files

- ◆ One message file (CATNIs) for each supported language.
- ◆ One CATRsc file for non message resources.



Student Notes:

Assigning Resources: Defined and Predefined

Using user defined resources

- These are messages that you define and use by yourself.



Using predefined resources

- Every dialog object (Panel, Toolbar, Workbench, ...) have some predefined resources (Title, Help, icon, ...).
- To use them, declare them in your CATNIs or CATRsc file.



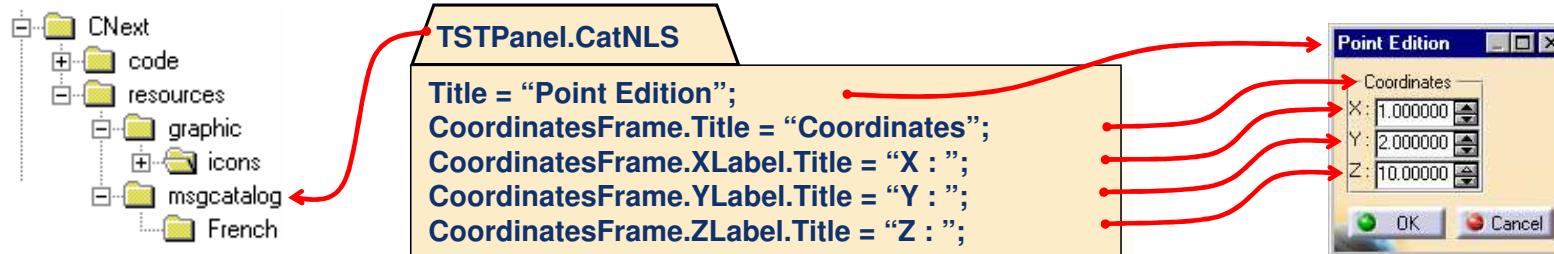
Student Notes:

Assigning Predefined Resources: Panel

- Concatenation
- Resources of contained objects can be defined in the container object resource file.
- Inheritance
- Derived object inherit from any resources defined in the father object.

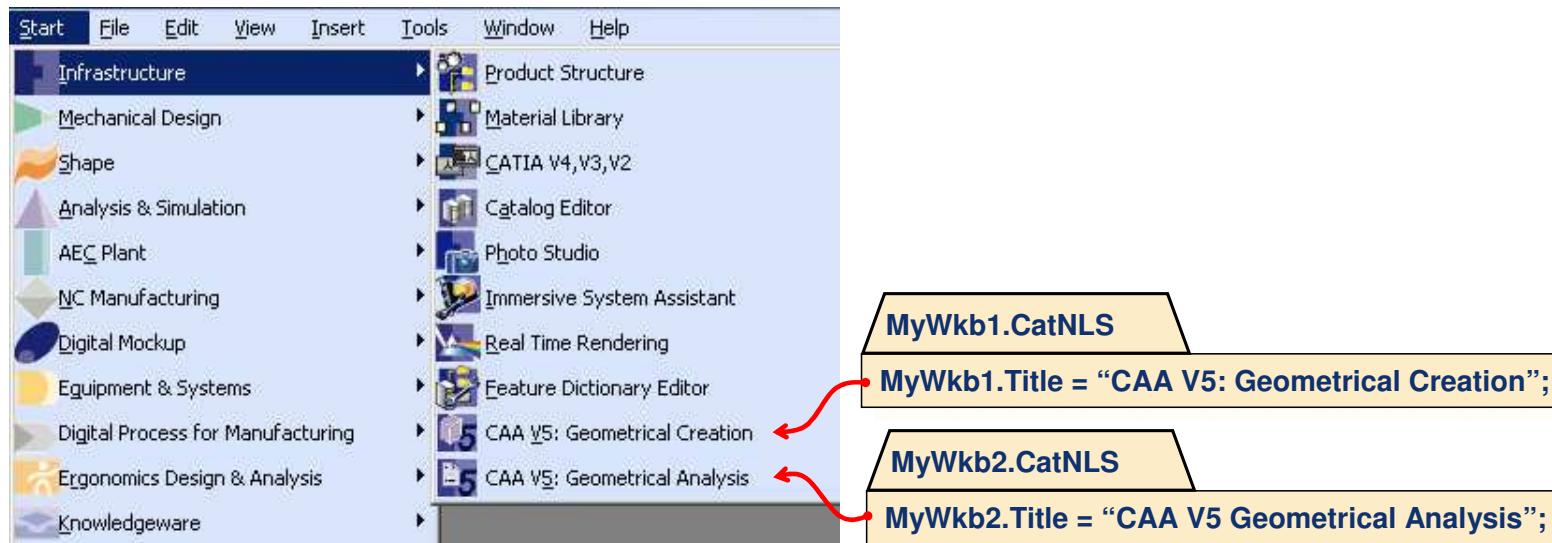
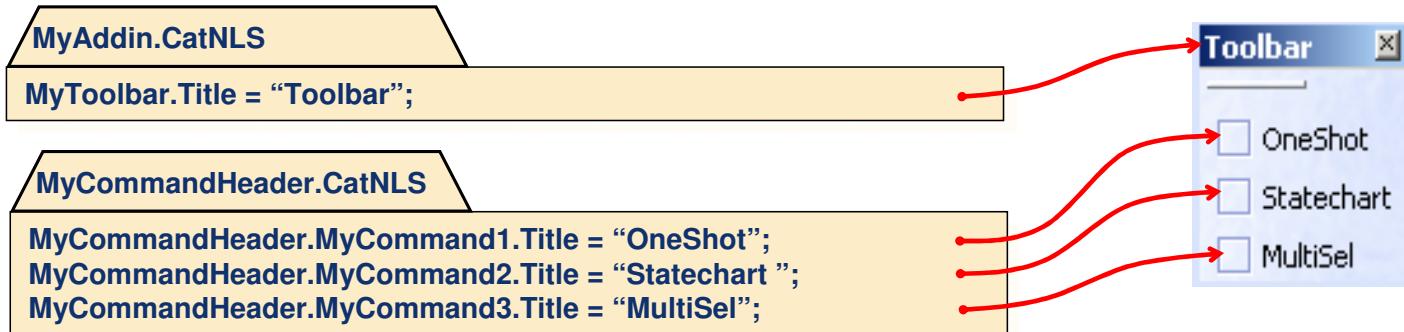
TSTPanel.cpp

```
void TSTPanel::Build()
{
    CATDIgFrame* pFrame = new CATDIgFrame(this, "CoordinatesFrame");
    CATDIgLabel* pLabel1 = new CATDIgLabel(pFrame , "XLabel");
    CATDIgLabel* pLabel2 = new CATDIgLabel(pFrame , "YLabel");
    CATDIgLabel* pLabel3 = new CATDIgLabel(pFrame , "ZLabel");
}
```



Student Notes:

Assigning Predefined Resources: Toolbar and Workbench



Student Notes:

CATDialog Predefined Resources

- There are several kind of Text and Icon Resources defined by several methods of CATDialog class
- **Text Resources**
 - ◆ Title : object title string
 - ◆ Mnemonic : Alt key shortcut to select a displayed menu or menu item
 - ◆ Accelerator : Ctrl key shortcut to select a menu item
 - ◆ Help : help message associated with the object
 - ◆ ShortHelp : short message displayed when the mouse stays over this object.
 - ◆ LongHelp : message displayed when the user clicks on Help button or on the ? box, the cursor becoming a ?, and clicks on the object to get help about it.
- **Icon Resources**
 - ◆ Icon : dialog object default icon
 - ◆ IconSel : icon displayed when this object is selected.
 - ◆ IconFocus : icon displayed when the mouse moves over this object.
 - ◆ IconDisabled : icon displayed when this object can not be selected.
 - ◆ IconType : type of the icon (Default, General, Creation, Modification, or Analysis), used to set a background color if the icon is transparent.

Exercise Presentation

→ Screw : Third part

And now practice one exercise, to learn about:

◆ Creating an interactive command :

- Create a panel
- Create a CATStatecommand
- Use Agent to select a sketch
- Create the screw head from the selected sketch

Student Notes:

Student Notes:

CAA V5 Visualization

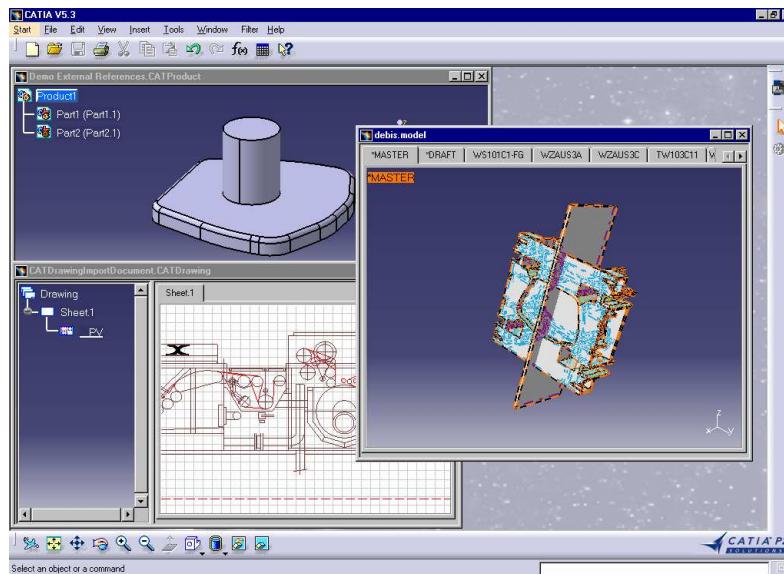
You will learn how to manage object representation

- Framework Objectives
- Model / View / Controller Architecture
- Main Visualization Interfaces

[Student Notes:](#)

Framework Objectives

- **Visualize a model**
 - ◆ A model is visualized thanks to a graphic representation
- **Multi-window management**
 - ◆ Components can be seen in several viewers or in several windows at the same time
- **Direct manipulation**
 - ◆ Viewers provide 3D and 2D manipulators to interact with the representation



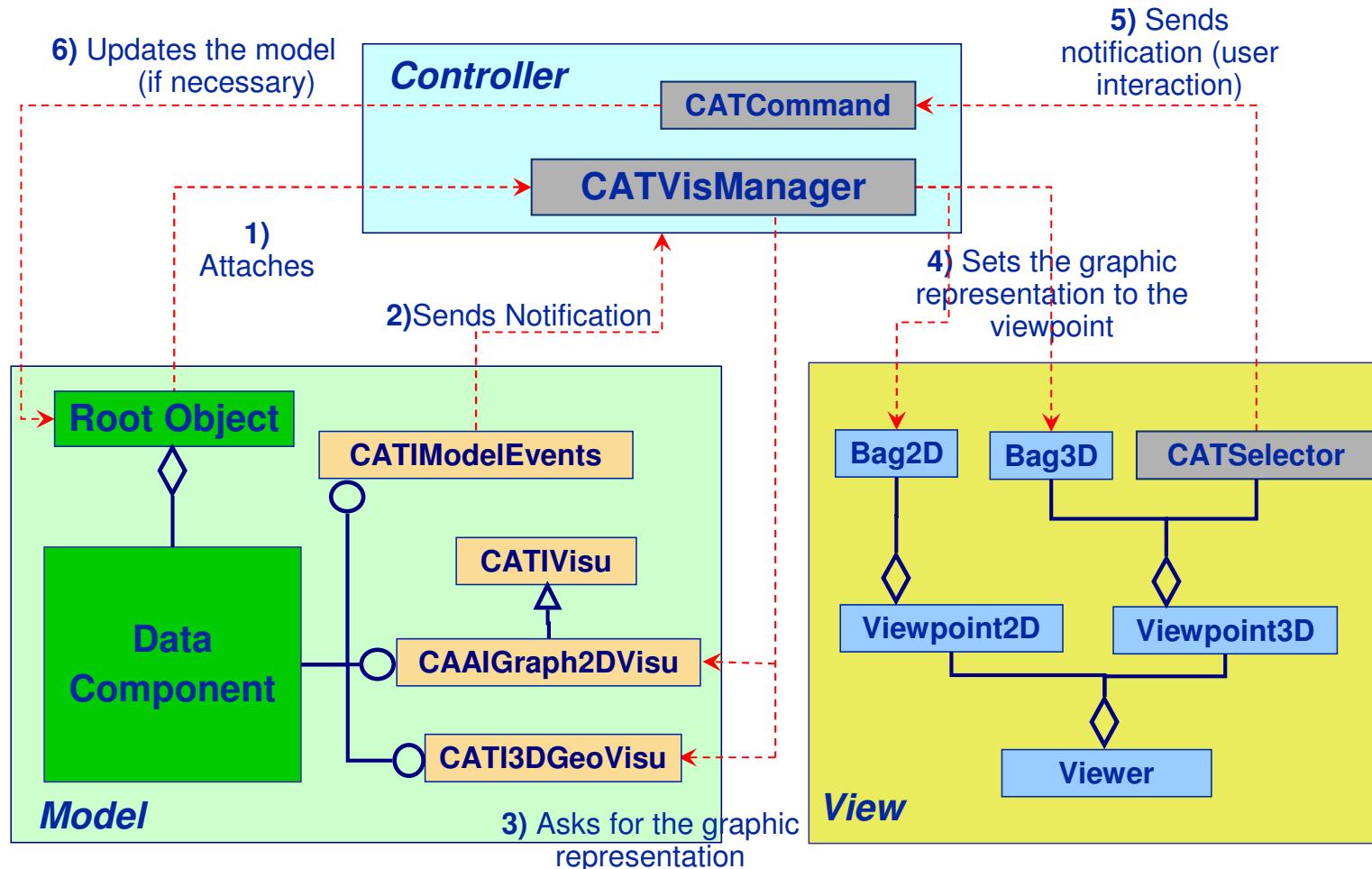
[Student Notes:](#)

Architecture Principles

- Model / View / Controller
 - ◆ Assume the independency between the data model and its representation (view)
 - ◆ Allow to change the viewer without impacting on the data
- Visualization is ruled by the controller
 - ◆ Manage one or several views of the data
 - ◆ Notify the views when the data change
 - ◆ Receive the user interactions to update the model when necessary (object creation, modification or deletion)

Student Notes:

Architecture Overview



Student Notes:

Controller Creation

- The manager is a unique instance of the **CATVisManager** class that manages the display of all the documents in all their windows
- The controller **CATVisManager** is unique
 - ◆ You are not responsible for creating or deleting it
 - ◆ You can retrieve it by a global access method :

```
CATVisManager::GetVisManager();
```

Student Notes:

Model: Graphic Properties

- Any features which are a graphic representation adheres to **CATIVisProperties** interface which allow to set some properties attributes with **SetPropertiesAtt()** method.
- To use this method, you have to precise three input argument:
 - ◆ The graphic properties with the **CATIVisPropertiesValues** class.
 - ◆ The type of property (**CATIVisPropertyType**)
 - **CATVPColor**: color.
 - **CATVPOpacity**: opacity.
 - ...
 - ◆ The type of geometry (**CATIVisGeomType**)
 - **CATVPMesh**: The geometry is surfacic.
 - **CATVPEdge**: The geometry is an edge.
 - ...

[Student Notes:](#)

Model: Refresh

- Sometimes it is necessary to force the refreshing of the specification tree and the 3D view to see the created feature:

◆ Refresh the specification tree: CATIRedrawEvent

```
CATISpecObject* piSOonRoot = piSpecOnMyObject->GetRootFather();
```

```
CATIRedrawEvent_var piRedrawOnRoot(piSOonRoot);  
piRedrawOnRoot->Redraw();
```

Retrieve the root father from a CATISpecObject interface on your object

Redraw the root father

◆ Refresh the 3D view: CATIModelEvents

```
CATISpecObject* piSOonRoot = piSpecOnMyObject->GetRootFather();
```

```
CATModify pModifyOnRoot(spSOonRoot);  
CATIModelEvents_var pMEOnRoot = spSOonRoot;  
pMEonRoot->Dispatch(pModifyOnRoot);
```

Retrieve the root father from a CATISpecObject interface on your object

Dispatch the notification

Student Notes:

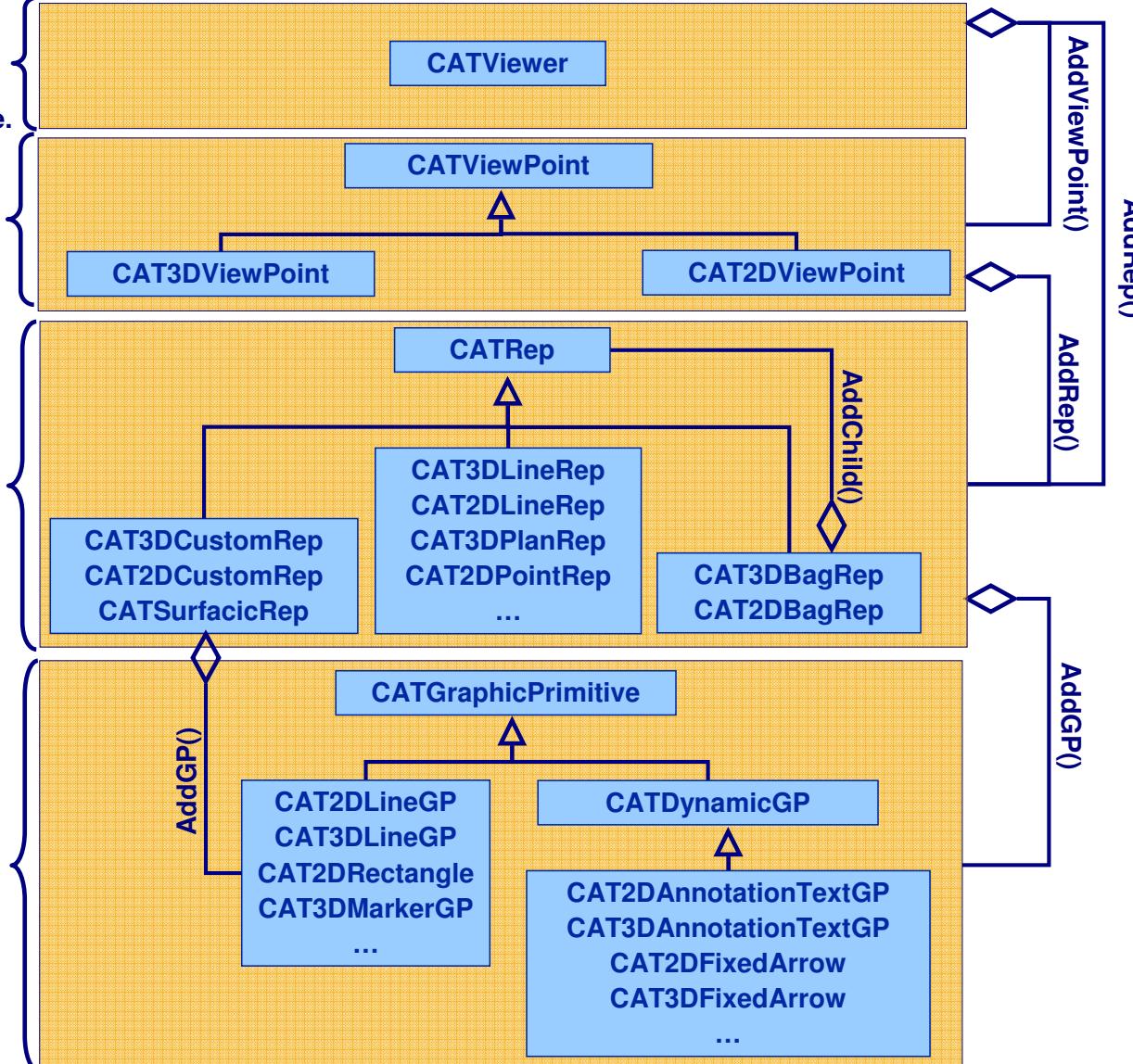
View: architecture

VIEWER:
Object allowing to navigate
inside the visualized model and
to define the visualization mode.

VIEW POINT:
View point of the user on
the model

REPRESENTATION:
Define the positioning and
the graphic attributes

GRAPHIC PRIMITIVE:
Define the geometry
Visualized.



Student Notes:

View: Creating Visualization Sample

Creation of an green arrow

```
CATMathPoint MPOrigin(0., 0., 0.); CATMathDirection Direction(1., 0., 0.); int Length = 20; int HeadHeight = 5;  
CAT3DFixedArrowGP* pArrow = new CAT3DFixedArrowGP(MPOrigin, Direction, Length, HeadHeight);
```

CATGraphicAttributeSet pGraphicAttributeSet;

```
pGraphicAttributeSet.SetColor(TRUECOLOR);  
pGraphicAttributeSet.SetColorRGBA(0, 255, 0);
```

```
CAT3DCustomRep* p3DCustomRep = new CAT3DCustomRep(pArrow, pGraphicAttributeSet);
```

CATMathPointf MPfOrigin(MPOrigin);

```
CAT3DBoundingSphere BSon3DCustomRep(MPfOrigin, 0, Length);  
p3DCustomRep->SetBoundingElement(BSon3DCustomRep);
```

```
_p3DBagRep = new CAT3DBagRep(); _p3DBagRep->AddChild(*p3DCustomRep);
```

Define the bounding visualization.

Creation of a 3D bag

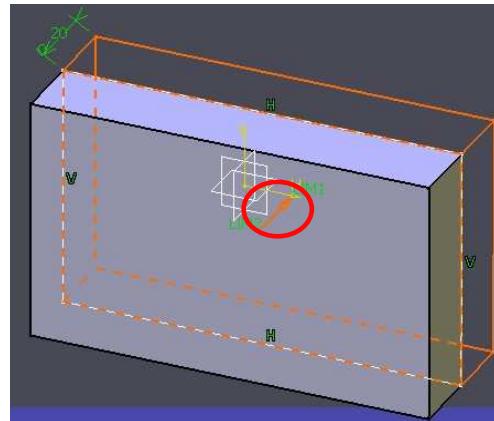
```
CATFrmLayout* pFrmLayout = CATFrmLayout::GetCurrentLayout();  
CATFrmWindow* pFrmWindow = pFrmLayout->GetCurrentWindow();  
CATViewer* pViewer = pFrmWindow->GetViewer();  
pViewer->AddRep(_p3DBagRep); pViewer->Draw();
```

Force the redrawing of the scene.

Student Notes:

View: manipulator sample (parameterization)

- The CATManipulator class provides the basic mechanisms to manipulate directly a visualized object.
- The following sample show:
 - how to parameterize a manipulator on the previous green arrow,
 - how to implement its displacement thanks to a callback.



```

_p3DBagRep = ...;



Type of displacement



_pManipulator = new CAT3DManipulator(this, "Manip", _p3DBagRep, CAT3DManipulator::DirectionTranslation);



CATMathPoint P(0., 0., 0.);



CATMathVector I(1., 0., 0.), J(0., 1., 0.), K(0., 0., 1.);



CATMathAxis Axis(P, I, J, K);



_pManipulator->SetPosition(Axis);



CATMathDirection D(0., 1., 0.); _pManipulator->SetTranslationDirection(D);



AddAnalyseNotificationCB(_pManipulator, CATManipulator::GetCATManipulator(),
(CATCommandMethod) &CAAMyCmd::CBManipulator, (void*) NULL);



Set the manipulator axis and direction



Define a callback to .manage the manipulator's displacement .


```

View: manipulator sample (Callback)

Student Notes:

```
void CAAMyCmd::CBManipulator(CATCommand* cmd, CATNotification* evt, CATCommandClientData data)
{
    CATMathAxis Position = ((CAT3DManipulator *)cmd)->GetPosition();
    CAT4x4Matrix InitialisationMatrix(Position);
    _p3DBagRep->SetMatrix(InitialisationMatrix);

    CATFrmLayout* pFrmLayout = CATFrmLayout::GetCurrentLayout();
    CATFrmWindow* pFrmWindow = pFrmLayout->GetCurrentWindow();
    CATViewer* pViewer = pFrmWindow->GetViewer();
    pViewer->Draw();
}
```

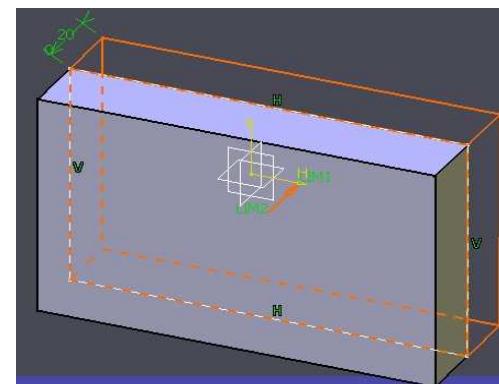
Set the new
bag position

Force the redrawing
of the scene.

Student Notes:

View: Temporary representation

- ❖ It is also possible to create temporary graphic representations of objects that are not part of the document, such as arrows to specify a direction of a face.
- ❖ These temporary representations are created independently of the displayed objects and then set to them.
- ❖ You can also derive from the **CATModelForRep3D** class:
 - ◆ to create your own component
 - ◆ to make it selectable in the 3D view
 - ◆ to interact with the view
(change arrow direction for example).
- ❖ These object has to be put inside the ISO (cf Application Frame chapter) to have a visualization.



View: Temporary representation (sample)

Student Notes:

```
CATMathPointf firstPoint (10.,10.,10.);  
CATMathPointf secondPoint (20.,20.,20.);  
CAT3DLineRep * pMyRep = new CAT3DLineRep(firstPoint,secondPoint);
```

Create a new rep

```
CATModelForRep3D * pFor3DModel = new CATModelForRep3D();  
pFor3DModel->SetRep(pMyRep);  
pMyRep = NULL;
```

Set the representation
to the new
CATModelForRep3D

```
CATFrmEditor * pEditor = CATFrmEditor::GetCurrentEditor();  
CATISO *pISO = pEditor-> GetISO( ) ;  
pISO->AddElement(pFor3DModel);
```

Use the ISO to
visualize the object

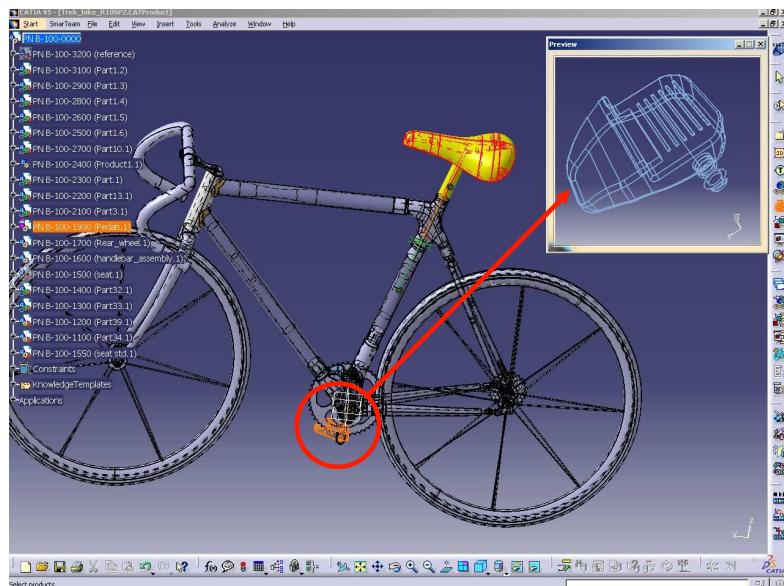
Student Notes:

View : Manage 2D-3D Viewers in Dialog Boxes

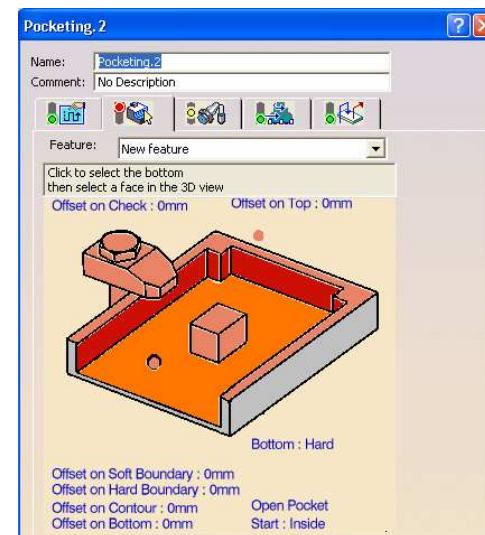
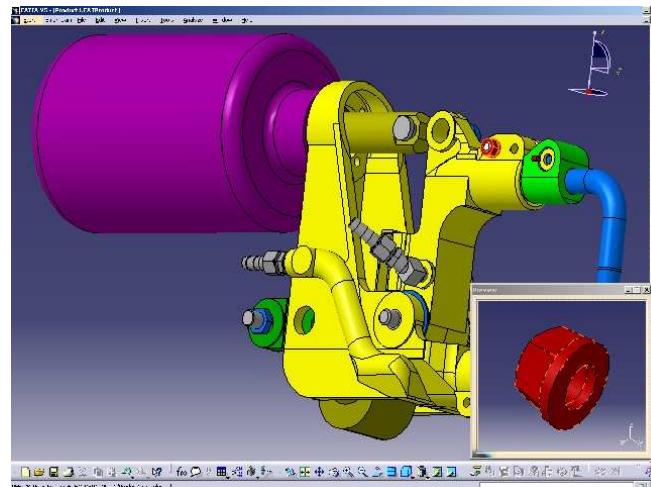
Added value of viewers in dialog boxes

Isolate an object

Simplify a view



Ease user interaction



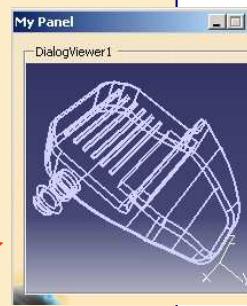
Student Notes:

View : Manage 2D-3D Viewers in Dialog Boxes

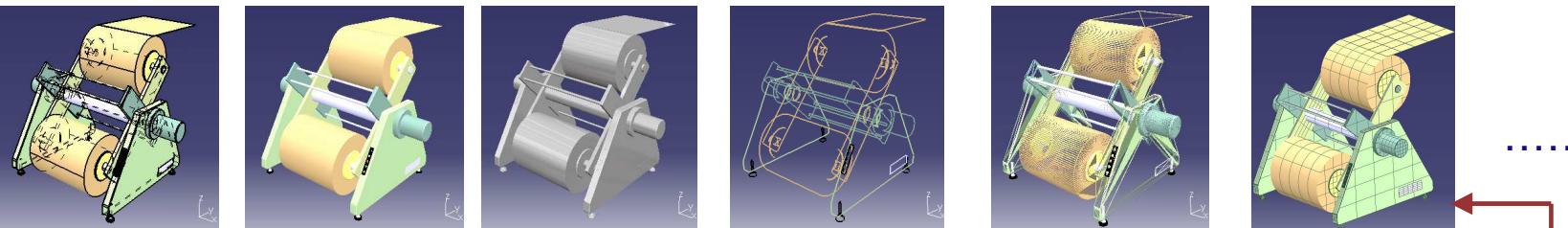
- Create a 3D navigation viewer (with options : scale, zoom ...) in a dialog box

```
_viewer1 = new CATNavigation3DViewer(_father, // Dialog father
                                     "DialogViewer1", // viewer name
                                     CATDlgFraNoTitle, // frame style
                                     200, // width
                                     200); // height

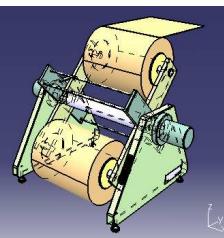
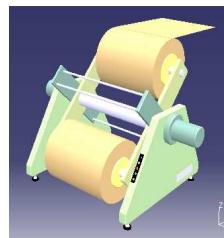
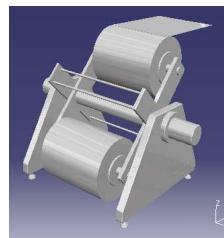
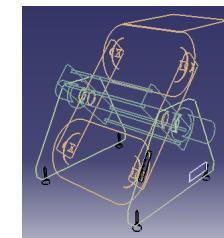
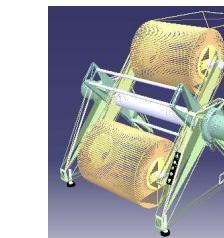
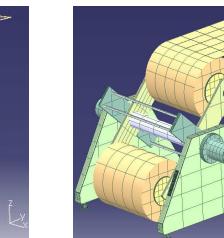
_viewer1->SetViewMode(0xFFFF,0); // initalize the view mode
_viewer1->SetViewMode(VIEW_EDGE , 1); // Change the CATViewModeType in Edge
```



- Different view mode types with CATViewModeType



Copyright DASSAULT SYSTEMES

					
Hidden edge (need : mesh+edge)	Mesh	Material (requires : mesh)	Outline	Polygon (requires : mesh)	Isopar

Enable isoparametrics generation
Number of isoparametrics in U and V

4

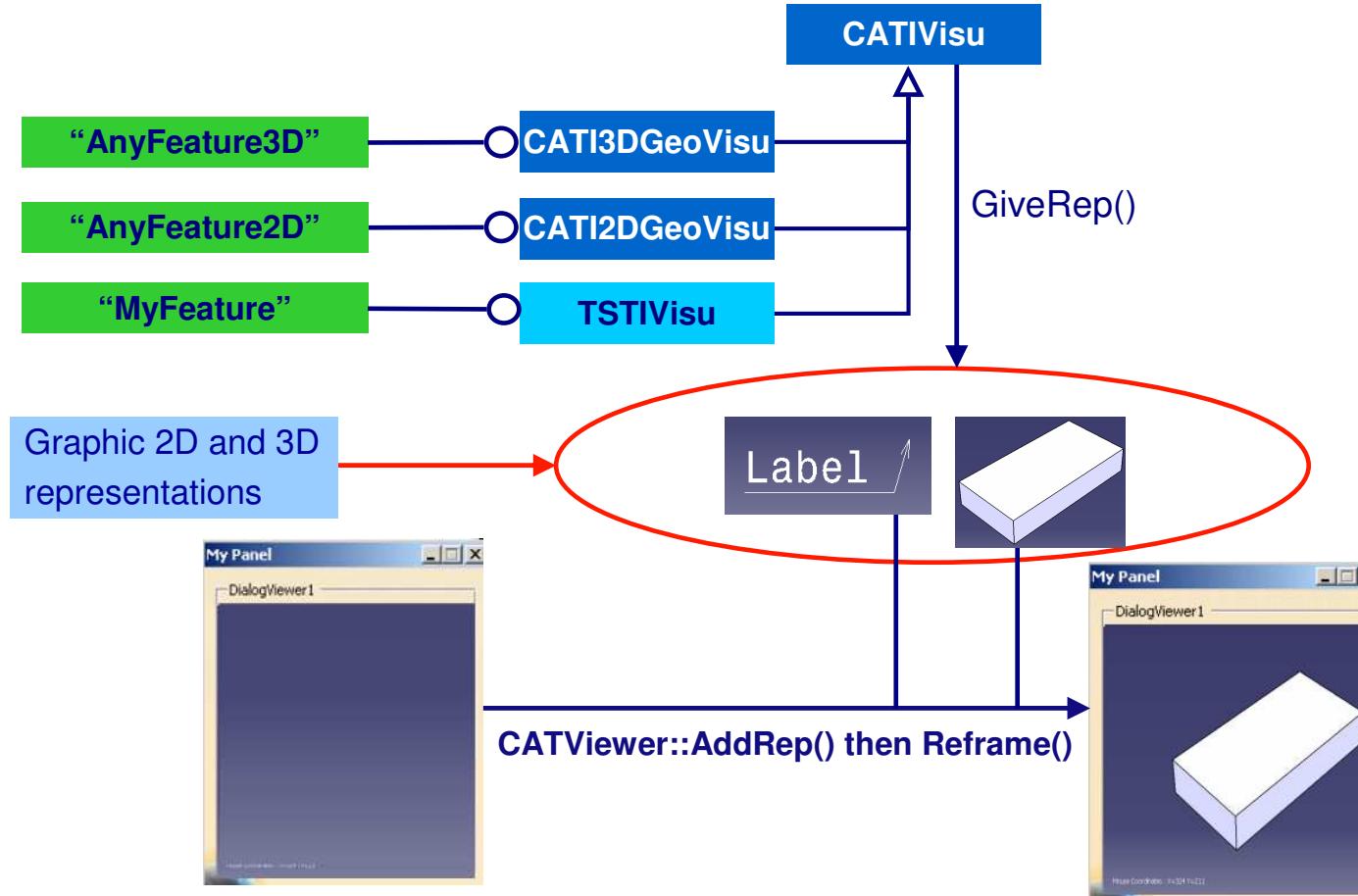
Tools/Options/Display/Performances

Restart Session

[Student Notes:](#)

View : Manage 2D-3D Viewers in Dialog Boxes

How to add graphic representations to your navigation viewer



Student Notes:

View : Manage 2D-3D Viewers in Dialog Boxes

- Manage interactions in your 3D navigation viewer

```
viewer->SetFeedbackMode(TRUE),  
::AddCallback(viewer,  
             CATViewer::VIEWER_FEEDBACK_UPDATE(),  
             (CATSubscriberMethod)&MyClass::MyCBMethod, NULL);
```

activates notifications when an interaction occurs
To receive these notifications

```
void MyClass::MyMethodCB(..., CATNotification * iNotification, ...)  
{  
    CATVisViewerFeedbackEvent * pFeedbackEvent = NULL ;  
    pFeedbackEvent = (CATVisViewerFeedbackEvent*) iNotification;  
  
    int context=pFeedbackEvent->GetContext();  
    If (context==Move)  
        .....  
    else if (context== Preactivate)  
        .....  
}
```

Treat the event depending on the context

Get the notification

Context can be :

- Preactivate
- MoveOver
- Move
- EndPreactivate
- BeginManipulate...

View : Manage 2D-3D Viewers in Dialog Boxes

Student Notes:

• Lifecycle management

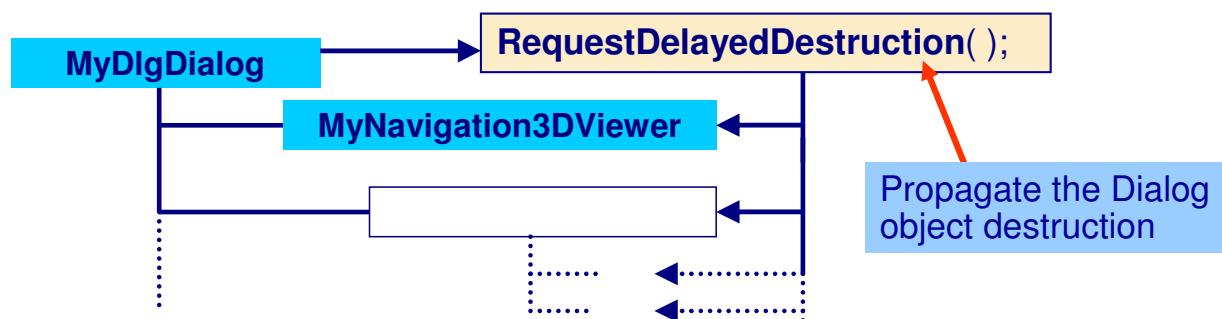
◆ Remove the representations from the viewer



◆ Destroy the graphic representations



◆ Destroy the dialog objects



Student Notes:

CAA V5 Geometric Modeler (CGM)

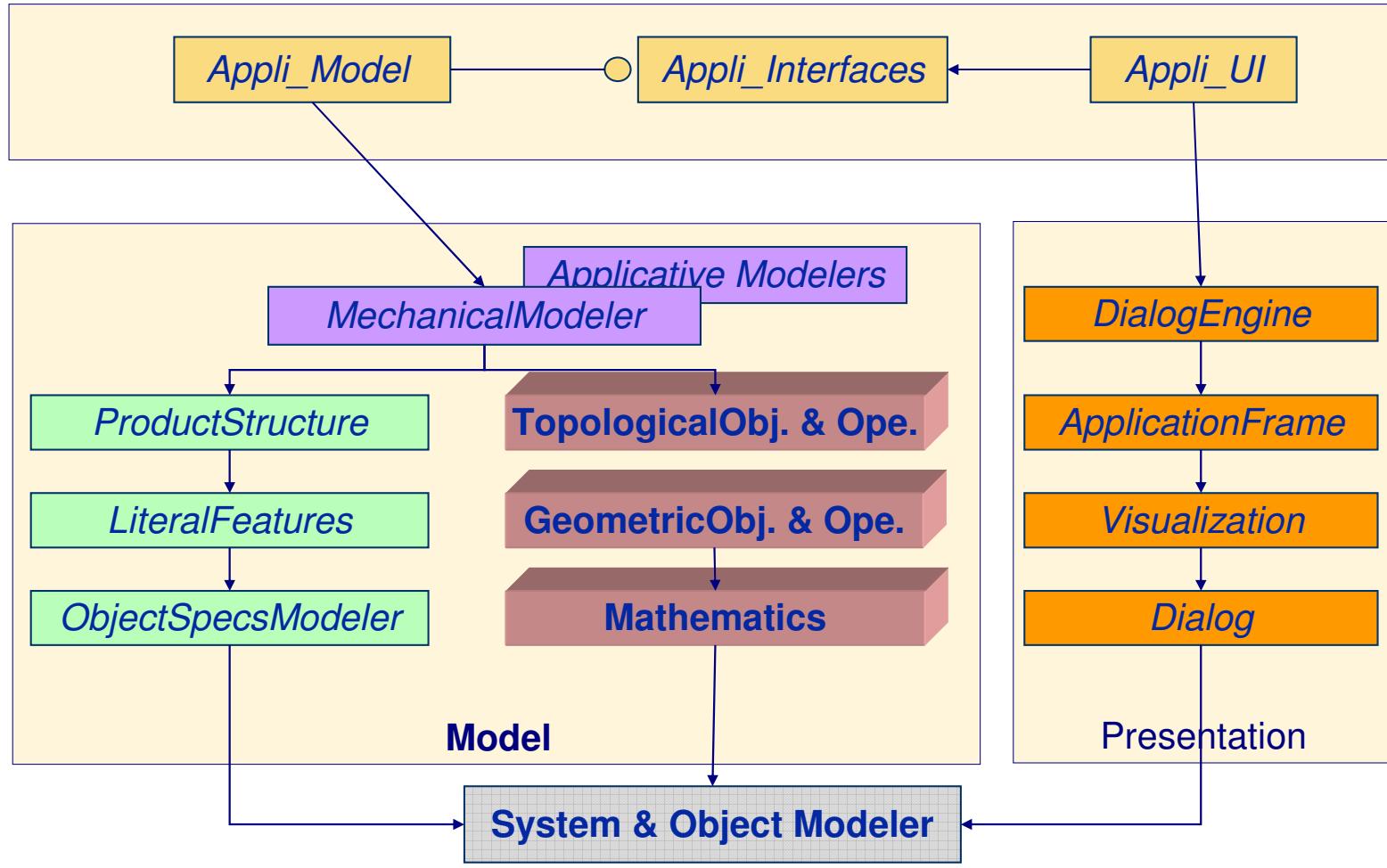
You will learn what are the CATIA geometric and topological capabilities, and how to manipulate them.

- Objectives
- Key technical features
- Mathematics & Advanced Mathematics
- Geometric Objects
- New Topological Objects

[Student Notes:](#)

CGM in the overall architecture

Applications (Part Design, Generative Shape Design, ...)



[Student Notes:](#)

Mathematics Overview

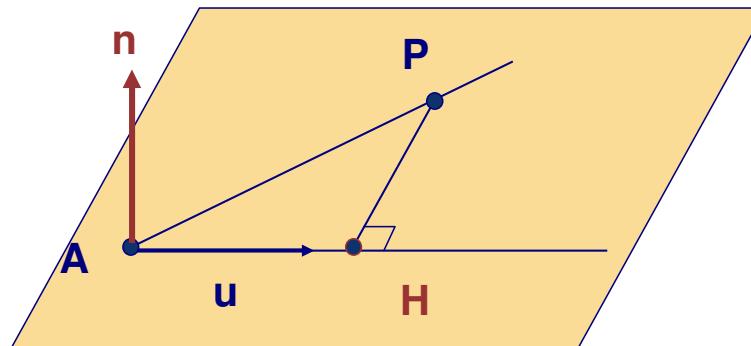
- Provides basic mathematical classes
 - ◆ Point, Vector, Line, Plane, Circle, 2D and 3D Matrices, Transformations, Boxes, ...
- Widely used in operations offered by other CGM Components
- Independent of the CGM context
- Non streamable objects
- Optimized classes
- Mathematics-like code writing
 - ◆ use of strong typing and operator overloading
 - ◆ ease of use

[Student Notes:](#)

Mathematics-like code writing

- Computation of the projection of the point P on the line (A,u)
- Computation of the normal of two vectors

```
CATMathVector u;  
CATMathPoint A,P,H;  
  
// Compute projection  
H = A + ( (P-A) * u ) * u  
  
// Compute normal  
CATMathVector n = u ^ (P-A);  
n.Normalize();
```



[Student Notes:](#)

Advanced Mathematics Overview

- Provides generic solvers dedicated to applications which want to do intensive mathematical computations
 - ◆ Implicit equation solver
 - ◆ Differential system solver
 - ◆ Non linear system solver
 - ◆ Function sampling and sampled points interpolation
- Provides the objects necessary to communicate with the solvers
 - ◆ Implicit
 - ◆ Functions of one or two variables and more
 - ◆ Rectangular matrices and linear systems
 - ◆ Intervals
- The geometric operators are obtained by derivation and/or aggregation of these generic solvers

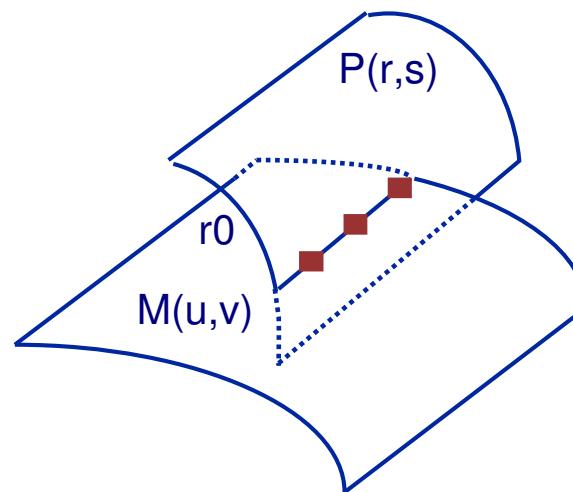
Student Notes:

Advanced Mathematics Usage in Geometric Operations

Mathematic algorithms are used in geometric operations:

Ex: intersection between two surfaces

- 1. curve / surface intersection
 - 3 non linear equations of 3 unknowns
 - $M(u,v) - P(r_0,s) = F(u,v,s) = 0$
- 2. look for singular points
 - 4 non linear equations of 4 unknowns
- 3. marching algorithm
 - 4 unknowns, 3 equations
- 4. create the curve
 - interpolation between the marching points



[Student Notes:](#)

Geometric Objects Overview

- A wide set of interfaces for using geometric objects:
 - ◆ Lines, curves, conics, NURBS curves ...
 - ◆ Canonical surfaces, NURBS surfaces, revolution surface,...
 - ◆ Curves of any type on surfaces of any type
 - ◆ Laws
 - ◆ Capability to introduce foreign curve and surface definition
- General data management capability
 - ◆ Interface / implementation concept
 - ◆ Factory and geometric objects life cycle
 - ◆ Stream and unstream
 - ◆ Persistent identification through tag identifier
 - ◆ User attribute definition
- Operators to copy and transform geometry

[Student Notes:](#)

Geometry definition

- Mathematical definition of a portion of the space which can be:

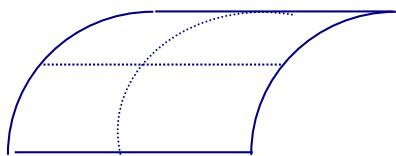
- ◆ a point:

-

- ◆ a curve:



- ◆ a surface:



- There is no visualization of the geometry in the 3D view of CATIA (it is just a mathematical definition).
- Geometry is streamed but it's a temporary object.

New Topological Objects Overview

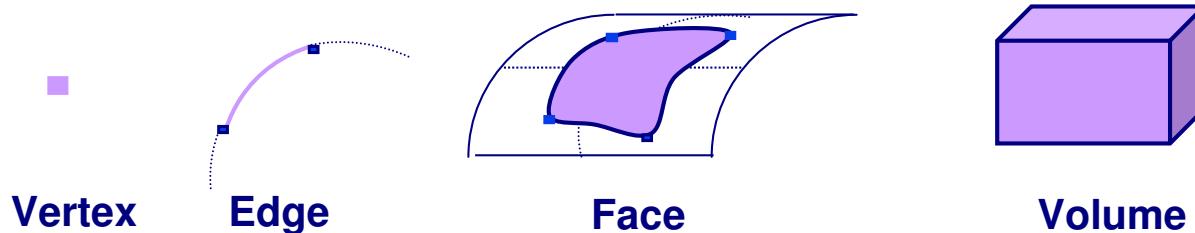
Student Notes:

- Abstract data model to describe basic topological relationships
- Object Definition
 - ◆ Enable basic creation of the topological objects
 - ◆ Navigation through any body definition
 - ◆ Support of non-manifold and multi-dimensional bodies
 - ◆ Minimal model size through data sharing at any level of the topological graph
- Independent of the underlying geometry

[Student Notes:](#)

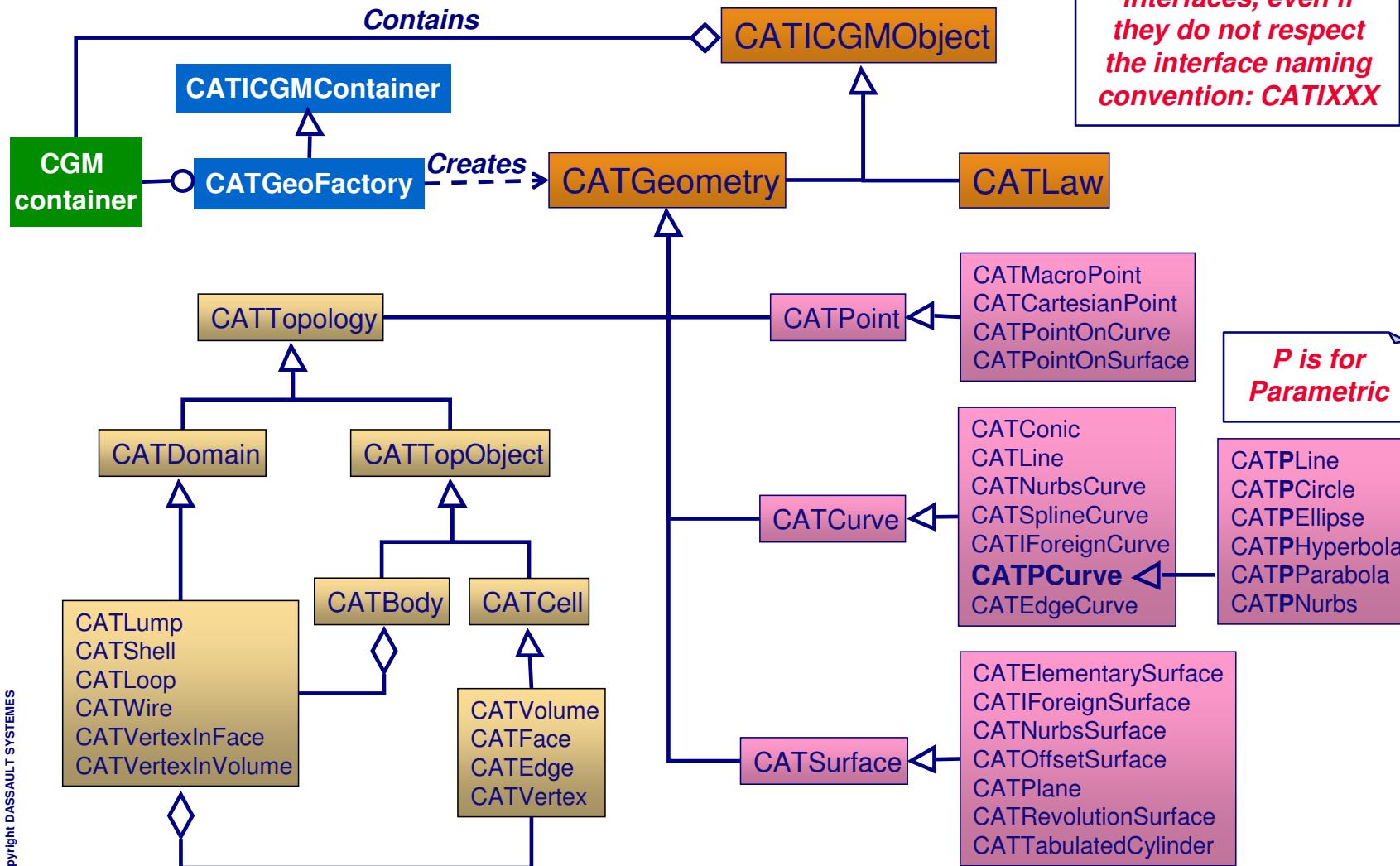
Topology Definition

- Topology is a building set for limiting the geometry (So, it's also a streamed and temporary object).
- Topology is visualized in the CATIA 3D view (contrary to geometry).
- The boundary dimension is lower than the limited geometry dimension.
- There's three kind of topological object:
 - Cell is a bounded region of an underlying geometry.
 - Cells of dim N-1 are grouped into Domains to bound cells of dim N.
 - A Body is a set of domains



Student Notes:

CGM Interface Hierarchy



Student Notes:

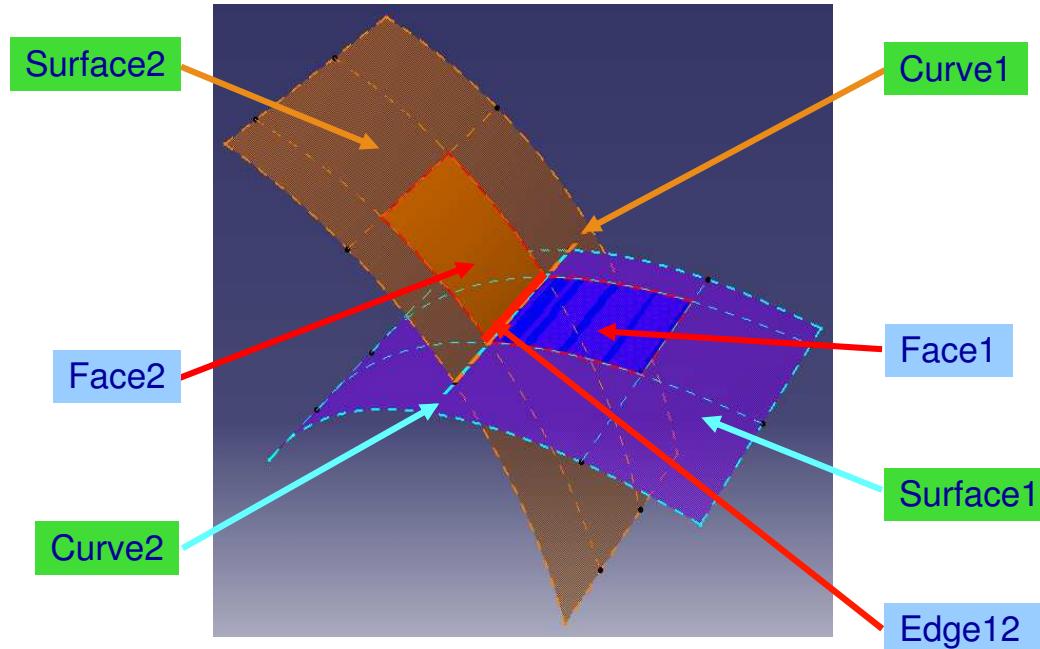
Conceptual Link between Topology & Geometry (1/2)

- According to the dimension of the topological object, the associated geometry can be:
 - ◆ CATMacroPoint
 - ◆ CATEdgeCurve
 - ◆ CATSurface
- These objects aggregate one or several geometrical objects of the dimension.

[Student Notes:](#)

Conceptual Link between Topology & Geometry (2/2)

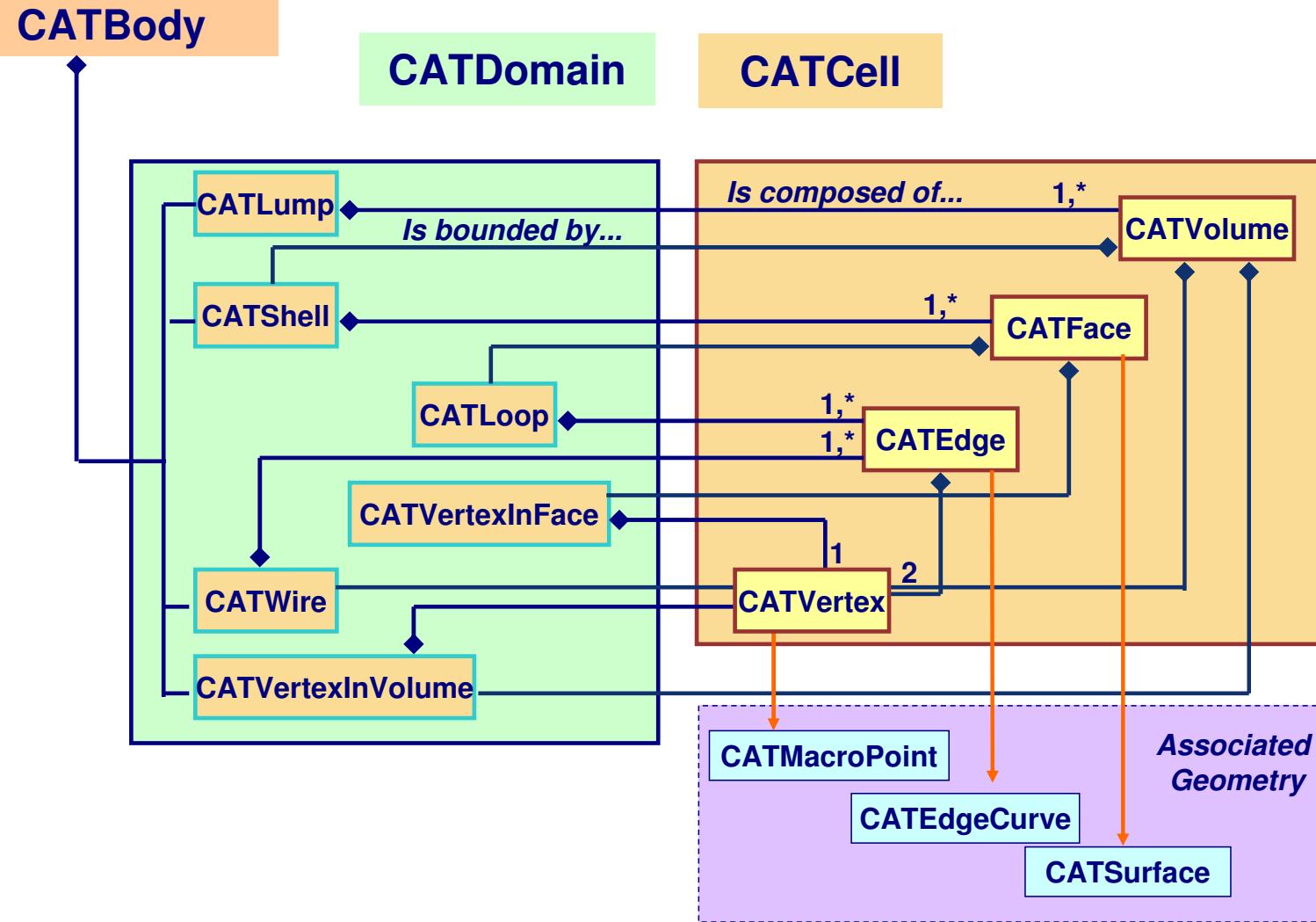
Topology
Geometry



- There is no visualization of the geometry (green) in the CATIA 3D view, only topology (blue).
- “Edge12” has an underlying “**edgecurve**” which aggregates two other curves (Curve1 and Curve2).

Student Notes:

Body, Cell and Domain Relationship



Student Notes:

The CATBody Interface

- Describe the concept of a Topological Body (restriction of the 3D geometrical space)
- Aggregate a set of non-necessarily connected Cells and refer (directly or indirectly) all the Cells needed for its construction
- Offer tools to navigate through itself
- Provide the factory to create all the Topological objects: CATCell(s) and CATDomain(s)

Student Notes:

Navigation through the Topological Data Structure

- The three main interfaces propose some navigation capabilities.
- At the CATBody level :
 - ◆ List of all the first level domains, hash table of all the cells of a given dimension, bounding edges, list of a set of faces, ...
- At the CATDomain level :
 - ◆ Recursive and non-recursive scans of the domain cells,...
- At the CATCell level :
 - ◆ Iterator of the bounding cells, search of adjacent cells,...

Student Notes:

Scanning a Topological Body: Domains

```
CATBody * piBody = ...;

// Get the number of domains for the body
long NumberOfDomains = piBody -> GetNbDomains();

// For each domain
for (int i = 1; i<= NumberOfDomains; i++)
{
    CATDomain * piDomain = piBody -> GetDomain(i);

    long tag = piDomain ->GetPersistentTag();           Get the domain's tag

    long NumberOfCells = piDomain ->GetNbCellUses();
    for (int j=1; j<= NumberOfCells; j++)
    {
        CATCell * piCell = piDomain ->GetCell(j);
        int dimCell = piCell ->GetDimension();
    }
}
```

Retrieve a **CATDomain** interface
on each domain of the body

You can scan the domain: retrieve
all cells and get their dimension.

Student Notes:

Scanning a Topological Body: Cells

```

// Retrieve a list of cells (faces)
CATLISTP(CATCell) CellList;
piBody->GetAllCells(CellList,2);

// Get the number of faces
int NumberOfFaces = CellList.Size();

// Loop over each face
for (int i=1; i< NumberOfFaces + 1; i++)
{
    CATGeometry * piGeomOfCell = CellList[i]->GetGeometry();

    // Compute the area of the face
    CATFace* piAFace = (CATFace*) (CellList[i]);
    double FaceArea = piAFace ->CalcArea();

    //Retrieve all planar surfaces
    CATSurface * piASurface = piAFace ->GetSurface();
    if (piASurface ->IsATypeOf(CATPlaneType))
        CATPlane * piPlanarSurface = (CATPlane*) piASurface ;

    long NumberOfDomainsForCell = CellList[i]->GetNbDomains();
    for (int k=1; k<NumberOfDomainsForCell; k++)
        CATDomain* Domain = CellList[i]->GetDomain(k);
}

```

Retrieve **all cells of dimension 2** (faces) in the body

Retrieve the **underlying geometry type** of the cell (ex: CATPoint, CATLine, ...)

Retrieve a **CATFace** interface on the cells to use the methods **CalcArea()** and **GetSurface()**

Use the method **IsATypeOf()** to test the type of a CGM object

Retrieve the aggregated domains to this cell

Student Notes:

CGM Objects rules

- In most cases, no AddRef/Release on a CGM interface
 - ◆ CGM objects don't follow typical CAA lifecycle rules
 - The reference counter is not incremented while object is returned by a function
 - Nevertheless an AddRef/Release error can produce a crash or memory leak
- In most cases, no QI needed
 - ◆ Behavior are mainly inherited
 - ◆ If ever a QI is done, don't forget to call Release
- CGM objects are deleted by the method Remove() implemented on CATICGMContainer
- Each time a CGM object is referenced by another one, an internal counter is incremented
 - ◆ The Remove() method is useless if the object to delete is referenced by an other CGM object

Exercise Presentation → Screw : Fourth part

And now practice one exercise, to learn about:

- ◆ Scan Geometry in Batch mode :
 - Creating a new batch program
 - Retrieve Using topological operators

Student Notes:

Student Notes:

CAA V5 Mechanical Modeler: Selection object

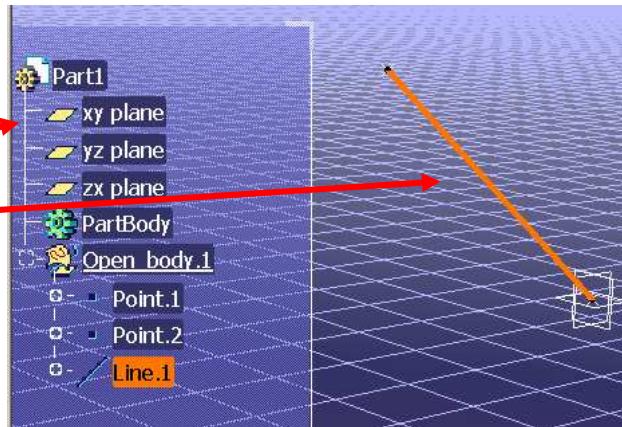
The Mechanical Modeler provide two different ways to select either a stable mechanical object (feature) or a temporary one (BRep access).

- Selection in an Interactive Session
- Selection Filter & Filter methods

[Student Notes:](#)

Selection in an Interactive Session

- ➊ You can select objects:
 - ◆ In the specification tree
 - ONLY features
 - ◆ In the 3D view
 - features or sub-elements



- ➋ Agents allow to select specific objects:

CATPathElementAgent



Acquisition agent that retrieves the Path (Part1/Open body.1/Line.1) of the selected object and stores it as its value.

CATFeatureAgent



Agent dedicated for mechanical feature selection.

CATFeatureImportAgent



Agent giving import capabilities: Design in context.

[Student Notes:](#)

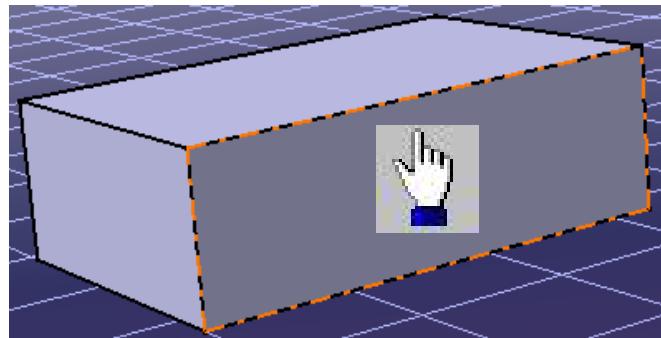
Filter to select a whole Feature

- You can select a feature in the 3D view or in the specification tree:
- Selection according to the type of features:
 - ◆ Surfacic and Wireframe features with GSMInterfaces interfaces like:
 - CATIGSMCircle
 - CATIGSMFill
 - ...
 - ◆ Solid features with PartInterfaces interfaces like:
 - CATIPad
 - CATIFillet
 - ...
- Selection according to the dimension of the topological result:
 - ◆ CATIMfZeroDimResult
 - ◆ CATIMfMonoDimResult
 - ◆ CATIMfBiDimResult
 - ◆ CATIMfTriDimResult
 - ◆ CATIMfInfiniteResult

Student Notes:

Filter to select a sub-element of a feature

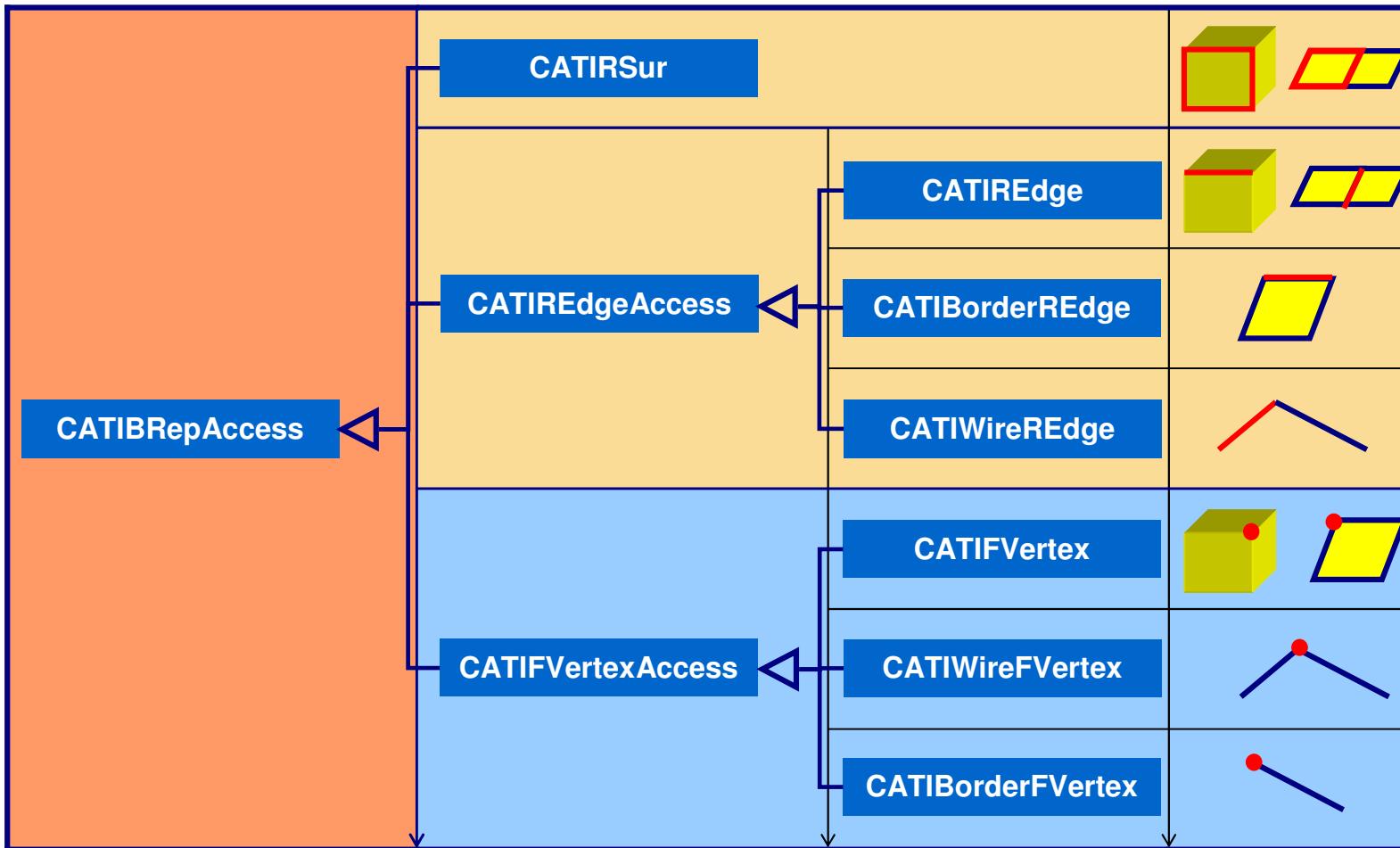
- A BRep Access is a TEMPORARY object built each time a sub-element is pre-selected or selected
 - ◆ it is a C++ object
- Thanks to a BRep Access filter, you can access to a topological element of a feature (cell)



- **CATBRepDecode()** allow you to create a Brep Access directly:
 - ◆ from a topological cell and a geometrical element support
 - ◆ from a topological body, a topological cell and the BRep Container
 - ◆ ...

[Student Notes:](#)

BRep Access Interfaces

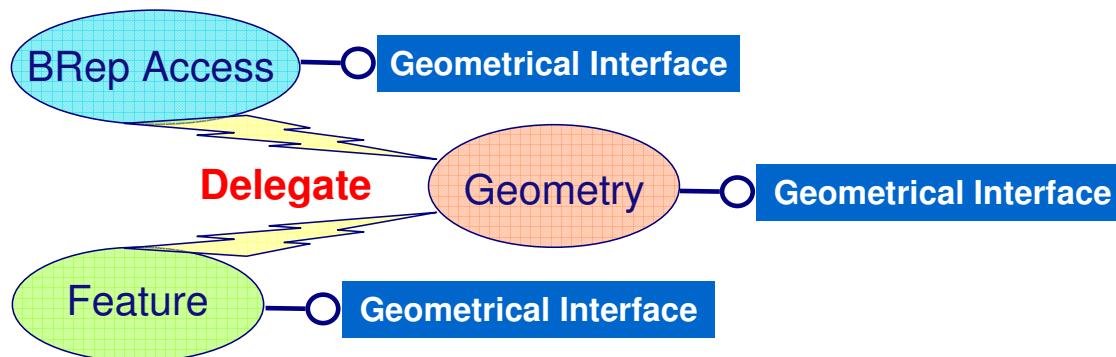


Student Notes:

Selection Filter: Geometrical and Topological interfaces

- Selection with Topological interfaces (ex: CATBody):
 - ◆ Impossible

- Selection with Geometrical interfaces (ex: CATPoint):
 - ◆ In the specification tree, the result will be a feature.
 - ◆ In the 3D view, the result will be a BRep Access.
 - ◆ This behavior is due to delegation (internal mechanism):



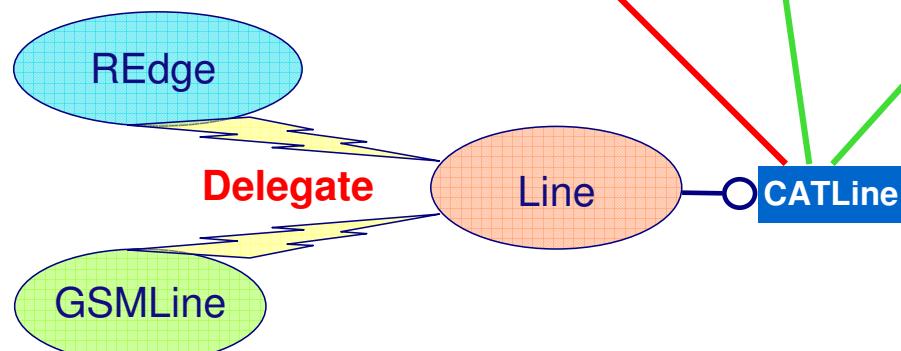
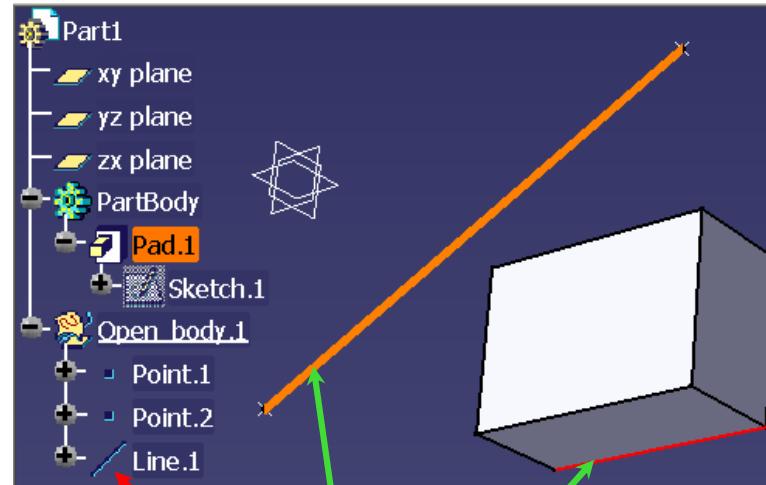
[Student Notes:](#)

Selection Filter: Delegation sample

Selection Filter: CATLine

→ The selection returns
a BRep Access Object

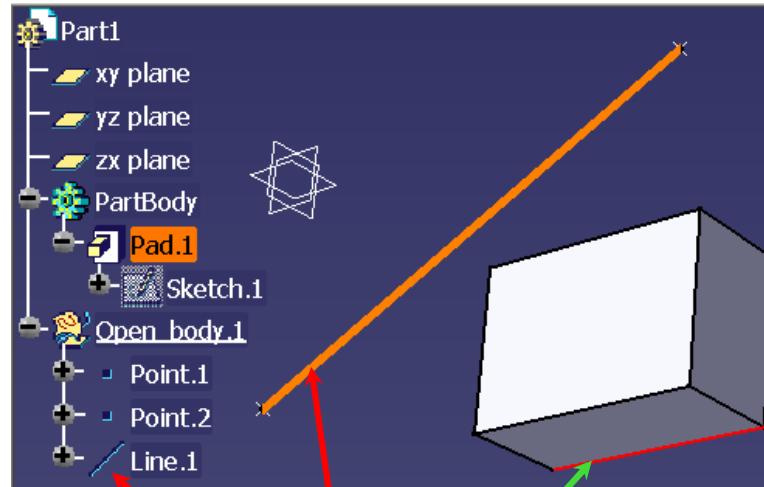
→ The selection returns
a Geometrical Feature



Selection Filter: Order

Student Notes:

- The selection returns a BRep Access Object
- The selection returns a Geometrical Feature

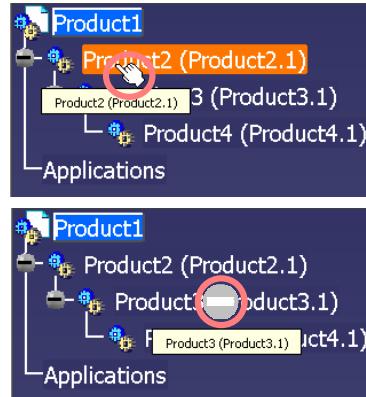


```
pAgent = new CAFeatureImportAgent ("Select");
CATListOfCATString Types;
Types.Append ("CATIGSMLine");
Types.Append ("CATLine");
pAgent->SetOrderedTypeList (Types);
```

Student Notes:

Filter Methods

- A selection filter allows you to select an object which adheres to this interface
- You can also use a filter method to define a more accurate selection



```
void CAACmd::BuildGraph()
{
    ...
    CATAcquisitionFilter* pFilter = Filter((FilterMethod )& CAACmd::Check, (void*) NULL);
    _pAgentSelect->SetFilter (pFilter);
    ....
}
```

Set the filter to the agent

```
CATBoolean CAACmd::Check(CATDialogAgent* iAgent, void* iData)
{
    CATIProduct* piProduct = ((CATPathElementAgent*)iAgent)→GetElementValue();
    if (IsFirstLevel (piProduct) ) return CATTrue;
    return CATFalse;
}
```

User method

[Student Notes:](#)

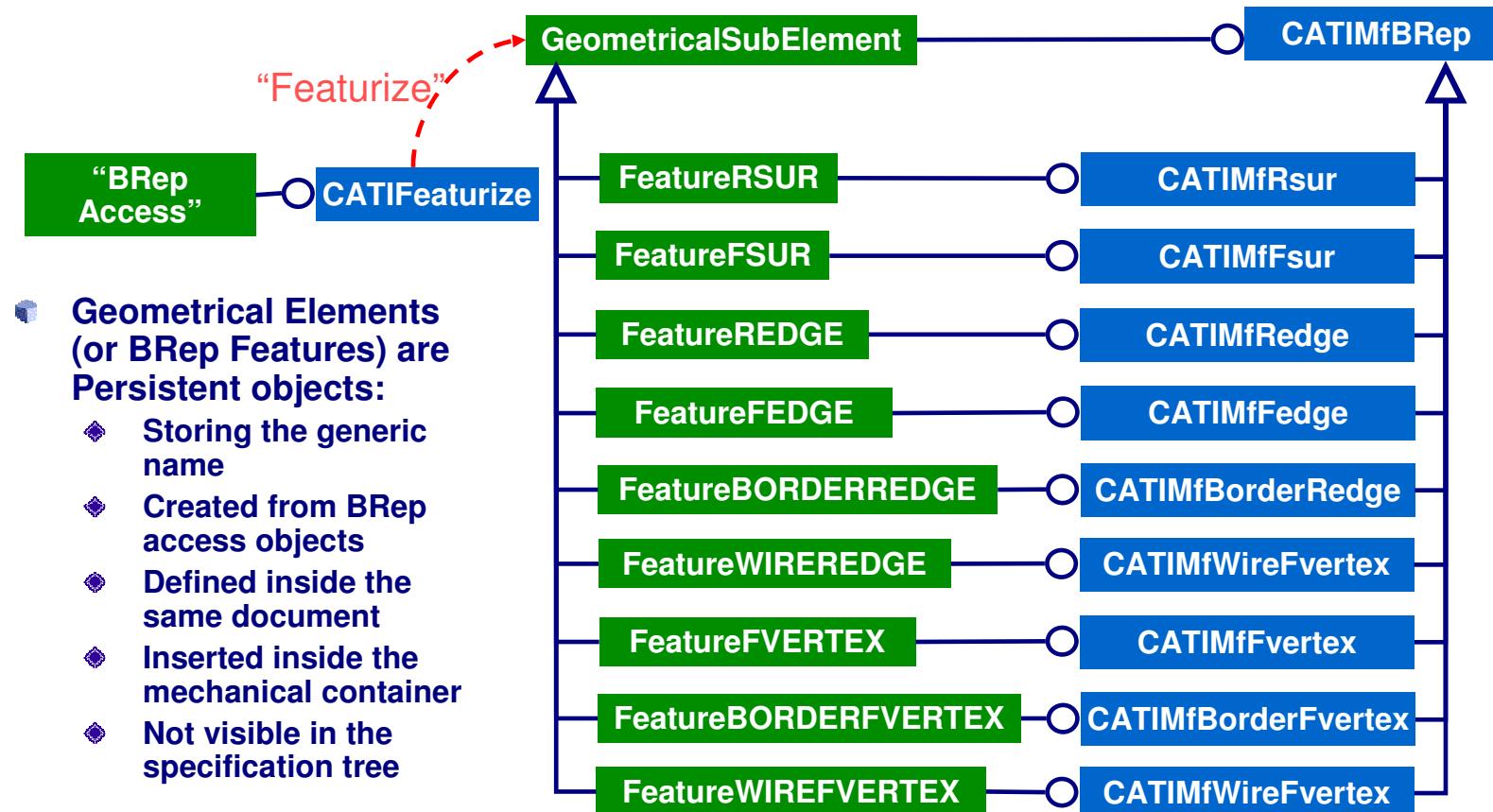
CAA V5 Mechanical Modeler: Topological stable access

The Mechanical Modeler provides a technique to access in a stable way to the sub-elements of a mechanical feature.

- **BRep Feature**
- **CATFeatureImportAgent**

[Student Notes:](#)

BRep Feature Introduction



- Geometrical Elements (or BRep Features) are Persistent objects:
 - ◆ Storing the generic name
 - ◆ Created from BRep access objects
 - ◆ Defined inside the same document
 - ◆ Inserted inside the mechanical container
 - ◆ Not visible in the specification tree

[Student Notes:](#)

Featurization Parameters (1/3)

◆ BRep Feature is defined by:

◆ its Support:

- Determines which feature is used to look for the cell

◆ its Featurization Mode

- Determines if the BRep feature should point on the re-limited or functional geometry

◆ its Build Type

- Determines if the BRep feature should be declared as a stable object from the generic name

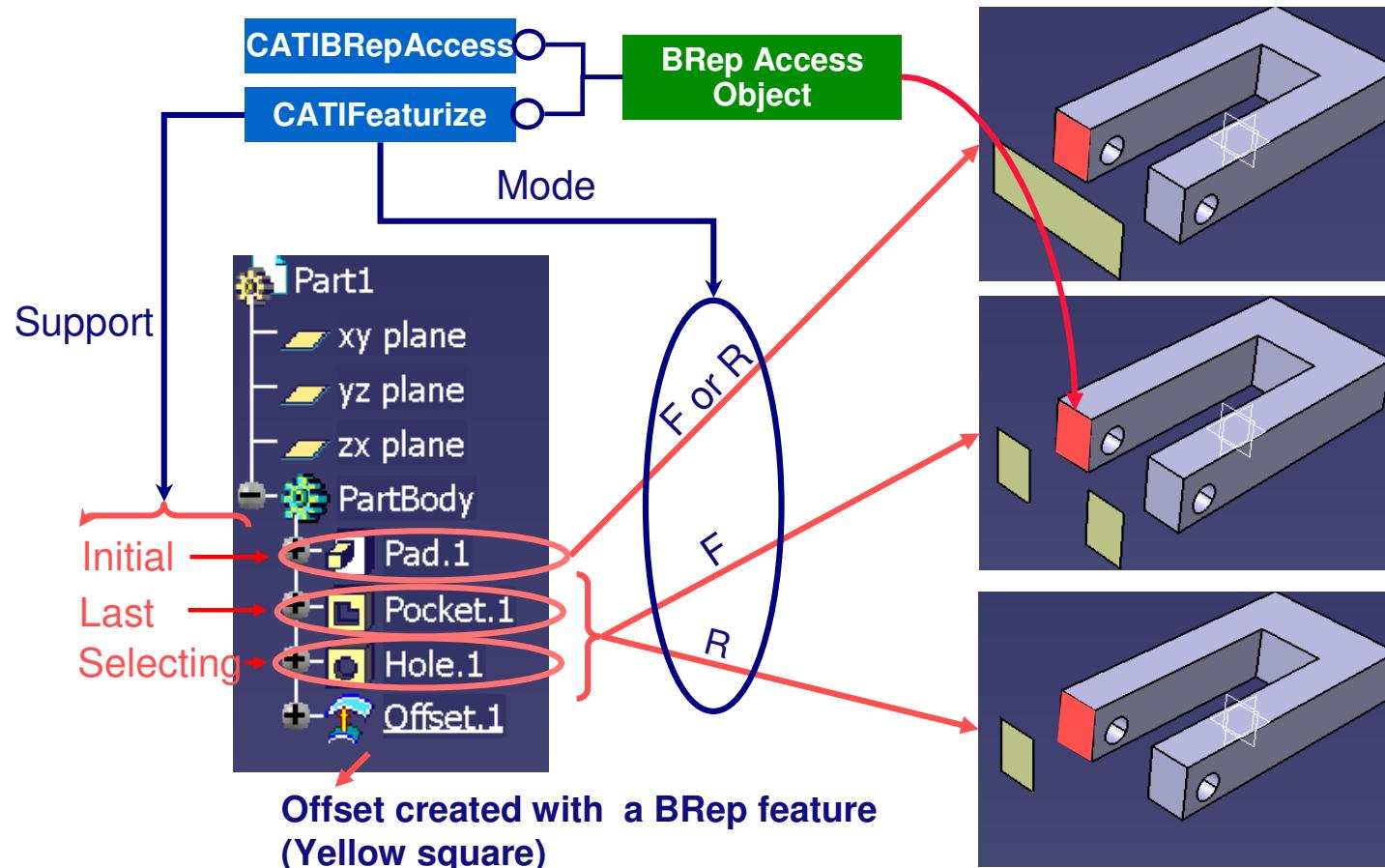
◆ its Duplication

- keep the default mode

◆ its Connexity

Student Notes:

Featurization Parameters (2/3)



[Student Notes:](#)

Featurization Parameters (3/3)

Build type:

- ◆ **MfTemporaryBody:**

- BRep feature will not be used in input for the procedural report

- ◆ **MfPermanentBody:**

- BRep feature will be used in input for the procedural report

- ◆ **In most cases,**

- MfPermanentBody is used for the feature which inherits from GSMGeom and MechanicalFormFeature
 - MfTemporaryBody is used for MechanicalContextualFeature

Duplicate type:

- ◆ **This option should not be set**

- ◆ **You have to keep the default mode MfNoDuplicateFeature**

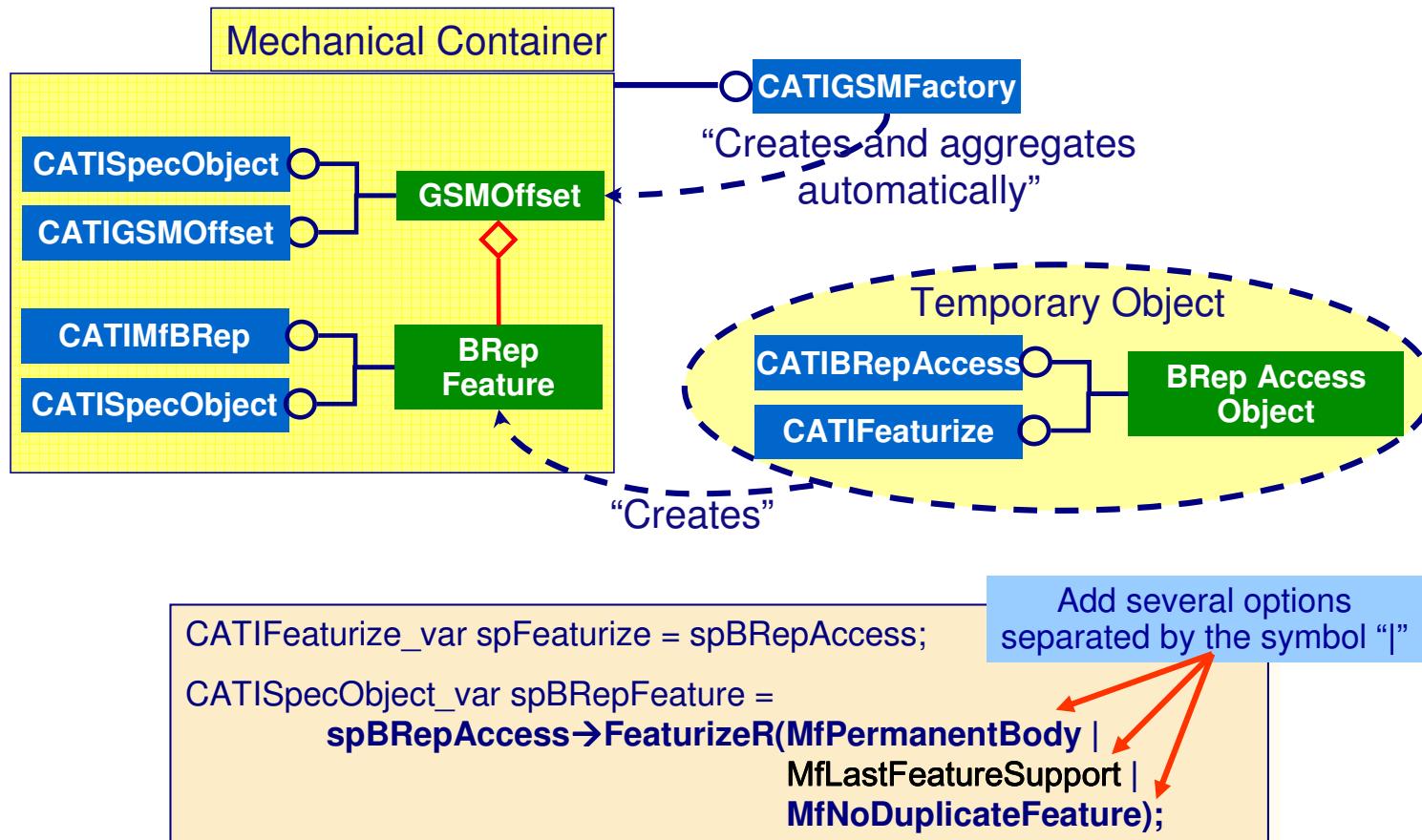
Connexity:

- ◆ **MfNoConnex**

- ◆ **MfConnex**

Student Notes:

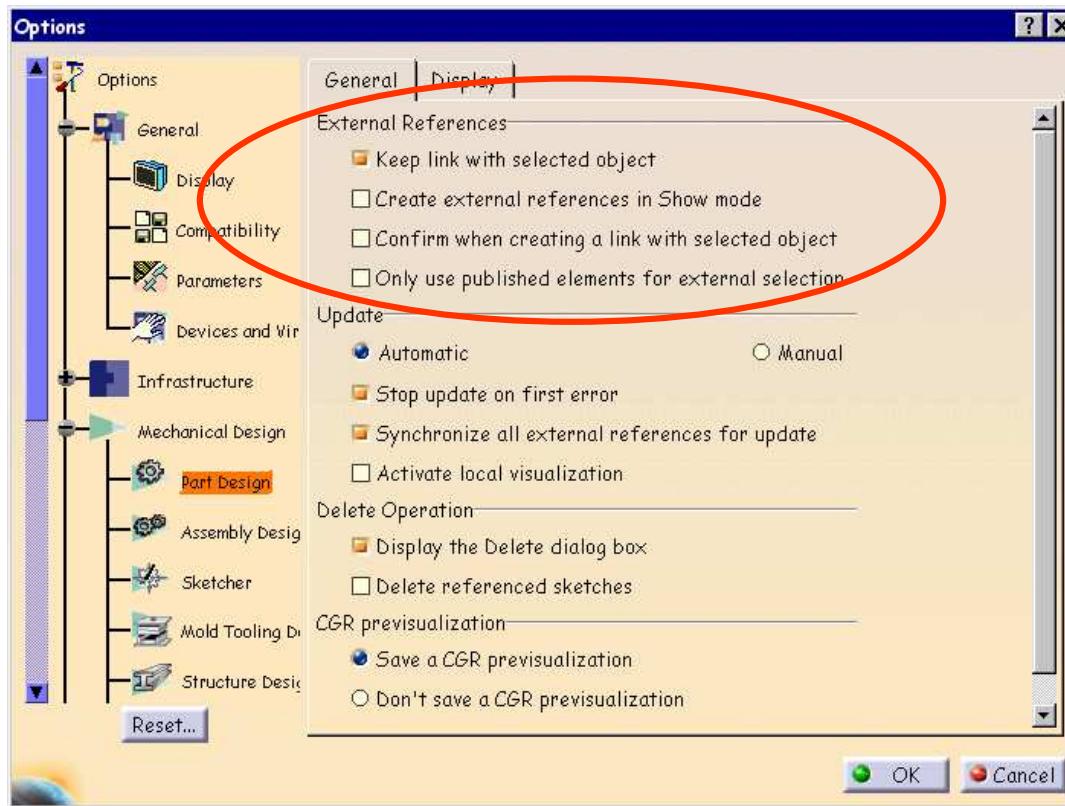
Featurization Implementation



Student Notes:

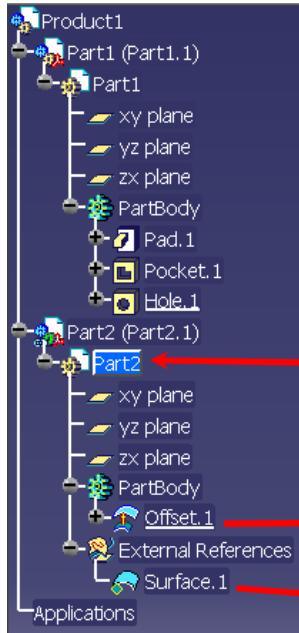
Featurization by CATFeatureImportAgent (1/2)

- This External Reference can be associative or not depending on the General Options of the Part Management.

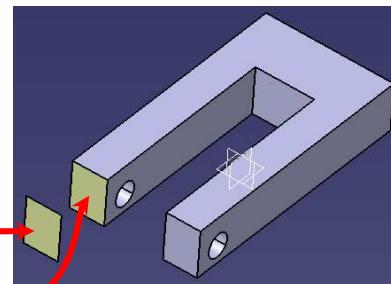


Student Notes:

Featurization by CATFeatureImportAgent (2/2)



- ❖ Agent which has two powerful properties:
 - ❖ Featurizes automatically the created BRep access (by the same previous options)
 - ❖ Creates an external link if needed



```
_pFeatureImportAgent->SetAgentBehavior (MfNoDuplicateFeature |  
MfPermanentBody |  
MfLastFeatureSupport |  
MfRelimitedFeaturization );  
...  
CATISpecObject_var spSO = _pAgentSelect->GetElementValue();
```

Exercise Presentation (Optional)

→ Screw : Fifth part

❖ And now practice one exercise, to learn about:

◆ Select sub-element in an interactive command :

- Create a Dialog Box
- Create a Callback
- Implement Callback

[Student Notes:](#)

Student Notes:

CAA V5 Knowledgeware: Parameters and Relations

You will learn what are the programming aspects around the Knowledgeware Products

- Overview on the CATIA V5 Knowledgeware Products
- Literal Features

[Student Notes:](#)

Knowledge Modeler (1/2)

- Dedicated to create and manipulate knowledge objects : parameters, formula or design table.
- Parameters are also called literal features.
- Provides :
 - ◆ basic features such as Length, Weight, ... with their corresponding units
 - ◆ More complex features.
- Allows to re-use parameters
 - ◆ Value management, units, range, tolerances.
 - ◆ Recognized by all the Knowledgeware tools.
- Allows also edition facilities.
 - ◆ Values edition.
 - ◆ Units compliancy checking.
 - ◆ Formulas edition.

[Student Notes:](#)

Knowledge Modeler (2/2)

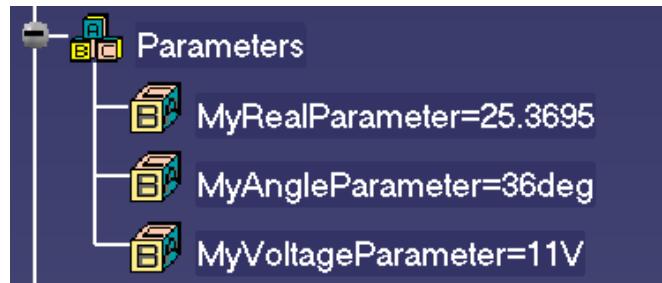
- Allows to create relations between Knowledgeware parameters
 - ◆ Different kind of relations
 - Design tables (based on Excel or text files).
 - Formula, rule, and check based on Knowledgeware language and on all types of parameters.
 - Sets of Equations
 - Set of predefined functions (mathematical, string, tables,...).
- Allows to create measures on features.
 - ◆ Set of provided measure functions (volume, cog, ...).
 - ◆ Integrated to the Knowledgeware language.
- Gives the ability to check design
 - ◆ Define and store sets of corporate rules.
 - ◆ Apply these rules to any design in order to check their compliancy to in-house standards.
- Services to manage parameters and relations are gathered in one framework : KnowledgeInterfaces.

[Student Notes:](#)

Literal Features (1/2)

The Literal Feature

- ◆ Models the value taken by a given instance of a given type.
 - ex: 3mm, "Hello world"
- ◆ A unit can be associated
- ◆ Used to be referenced within a Relation
 - ex: PartBody.1\Hole.1\Radius



The Literal Type

- ◆ Models a value type:
 - Length, Real, String, Boolean, Integer, ...

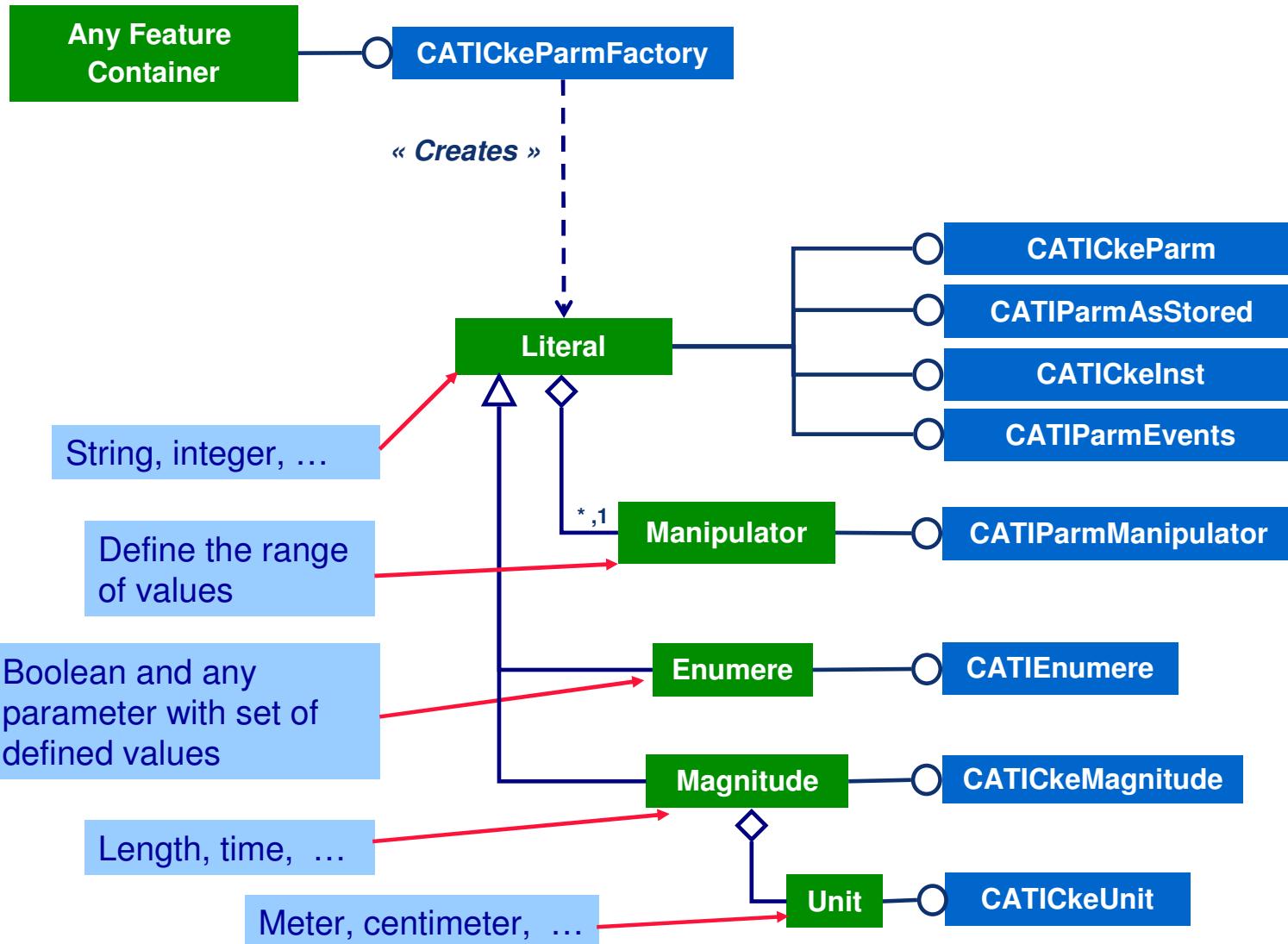
[Student Notes:](#)

Literal Features (2/2)

- Literal Features are created with the CATICkeParmFactory
 - ◆ Simple Type Parameters, such as Integer, real or String,
 - ◆ Dimension Type Parameters, such as Length or Angle,
 - ◆ Magnitude Type Parameters, such as dimensions
 - ◆ Enumerated Type Parameters
- Associated with one or several parameters, a manipulator gathers information about how this parameter can be modified, such as
 - ◆ the range of a parameter (min,max) ,
 - ◆ The step for the literal editor
- Such a manipulator is also created with the CATICkeParmFactory
- Literal features can then be used in relations.

Student Notes:

Literal Features Architecture



Student Notes:

Main Interfaces for Parameter and Value Management

- **CATICkeParm**
 - ◆ Parameter management in MKSA (meters-kilos-seconds-ampere) when dealing with values
 - **CATIParmAsStored**
 - ◆ Value management in model unit (mm and degree)
 - **CATICkeInst**
 - ◆ Value management (conversions)
 - **CATIParmEvents**
 - ◆ Add subscription to parameter events:
 - Changed Value
 - Renamed parameter
 - Changed Visualization when visibility changes, user access changes, driving relation is set or unset, activated or deactivated, current unit changes
 - Deleted parameter
 - **CATIParmManipulator**
 - ◆ Associated with one or several parameters
- CATICkeParm::SetManipulator (const CATIParmManipulator_var)**

Parameter code example

Student Notes:

```
CATICkeParmFactory *piParamFactory = NULL;  
hr = piSpecCont->QueryInterface( IID_CATICkeParmFactory,  
                                  (void **) &piParamFactory);  
  
// Create a Length type parameter: Radius  
// The radius is described by a length type parameter  
// Default unit is MKS unit i.e. millimeters  
  
double RadiusValue = 2.5;  
CATICkeParm_var spRadius = piParamFactory->CreateLength("Radius",  
                                         RadiusValue );  
  
CATICkeFunctionFactory_var FunctionFactory= NULL_var;  
FunctionFactory= CATCkeGlobalFunctions::GetFunctionFactory() ;  
FunctionFactory->AddToCurrentParameterSet( spRadius);
```

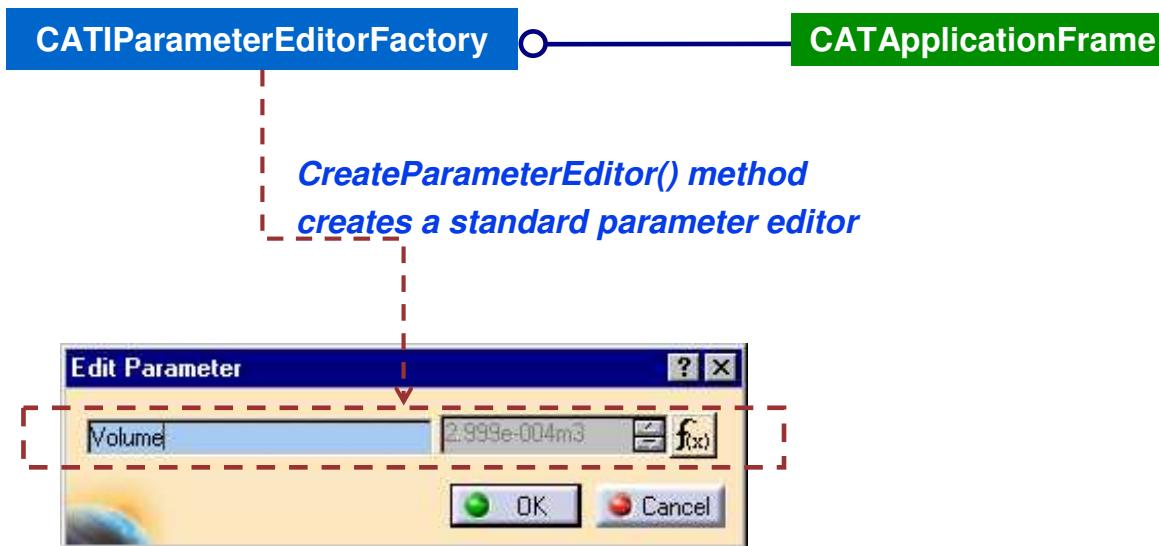


AddToCurrentParameterSet allows you to aggregate the parameter, so keep it persistent and so it can be seen in the specification tree

Student Notes:

Parameter Editor Management

- A Parameter Editor allows to modify the value of a parameter associated to it.
- It is created thanks to the **CATIParameterEditorFactory**.
- You can then attach the parameter editor to a parameter and customize its frame.



Student Notes:

Parameter Editor integration in a Panel

CreateManipulator: you must create a manipulator and associate it with the parameter

```
CATIParmManipulator_var spManipOnParm = piParamFactory->  
CreateParmManipulator();  
spManipOnParm ->SetAccurateRange (0.5,1,5,1);  
spRadius ->SetManipulator (spManipOnParm );
```

```
CATApplicationFrame * pAppFrame = CATApplicationFrame::GetFrame ();  
CATIParameterEditorFactory * piFactory = NULL;  
pAppFrame -> QueryInterface (IID_CATIParameterEditorFactory,(void **)&piFactory);
```

```
CATIParameterEditor* piEditor = NULL;  
piEditorFactory->CreateParameterEditor (iParent , "CAAParmEditor", 1, piEditor );
```

Create a standard parameter edition frame in a dialog

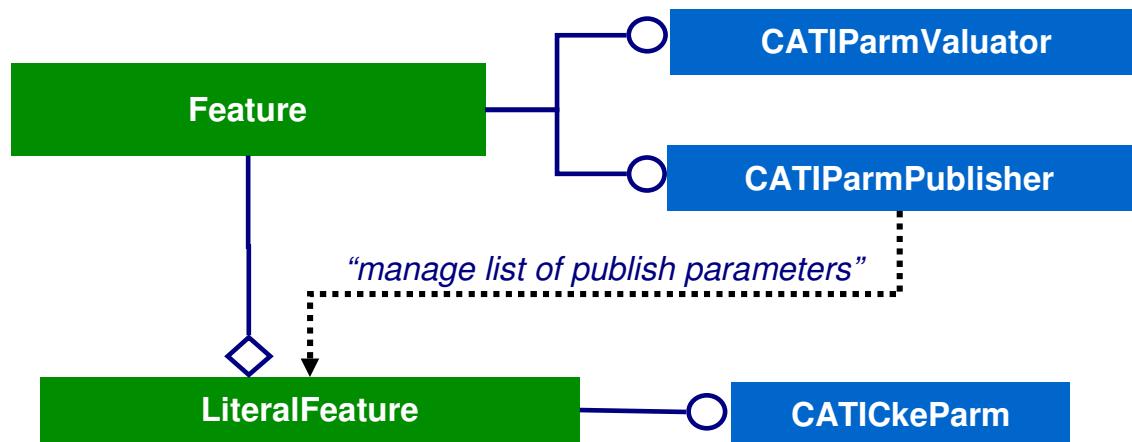
```
piEditor-> SetEditedParameter(spRadius );  
...
```

Set the parameter to be edited by this
parameter editor.

Student Notes:

Literal Features and Edition (1/2)

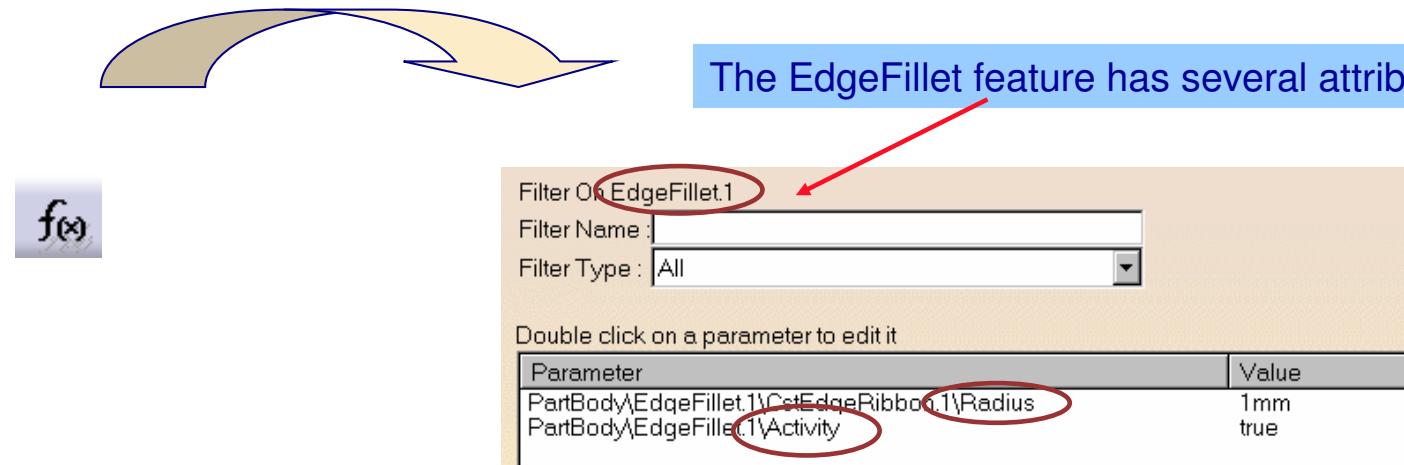
- When you click on the  icon, the characteristics of a feature are displayed in the parameters list.
- To benefit from this behavior, the feature and its parameters must fulfill two conditions:
 - The feature must implement CATIParmPublisher
 - Its attributes must be implemented as feature parameters



Literal Features and Edition (2/2)

Student Notes:

Example :



```
CATIParmPublisher_var spParmPub(spFillet);
CATLISTV(CATISpecObject_var) ListFound;
spParmPub->GetAllChildren( CATICkeParm::ClassName() , ListFound);
```

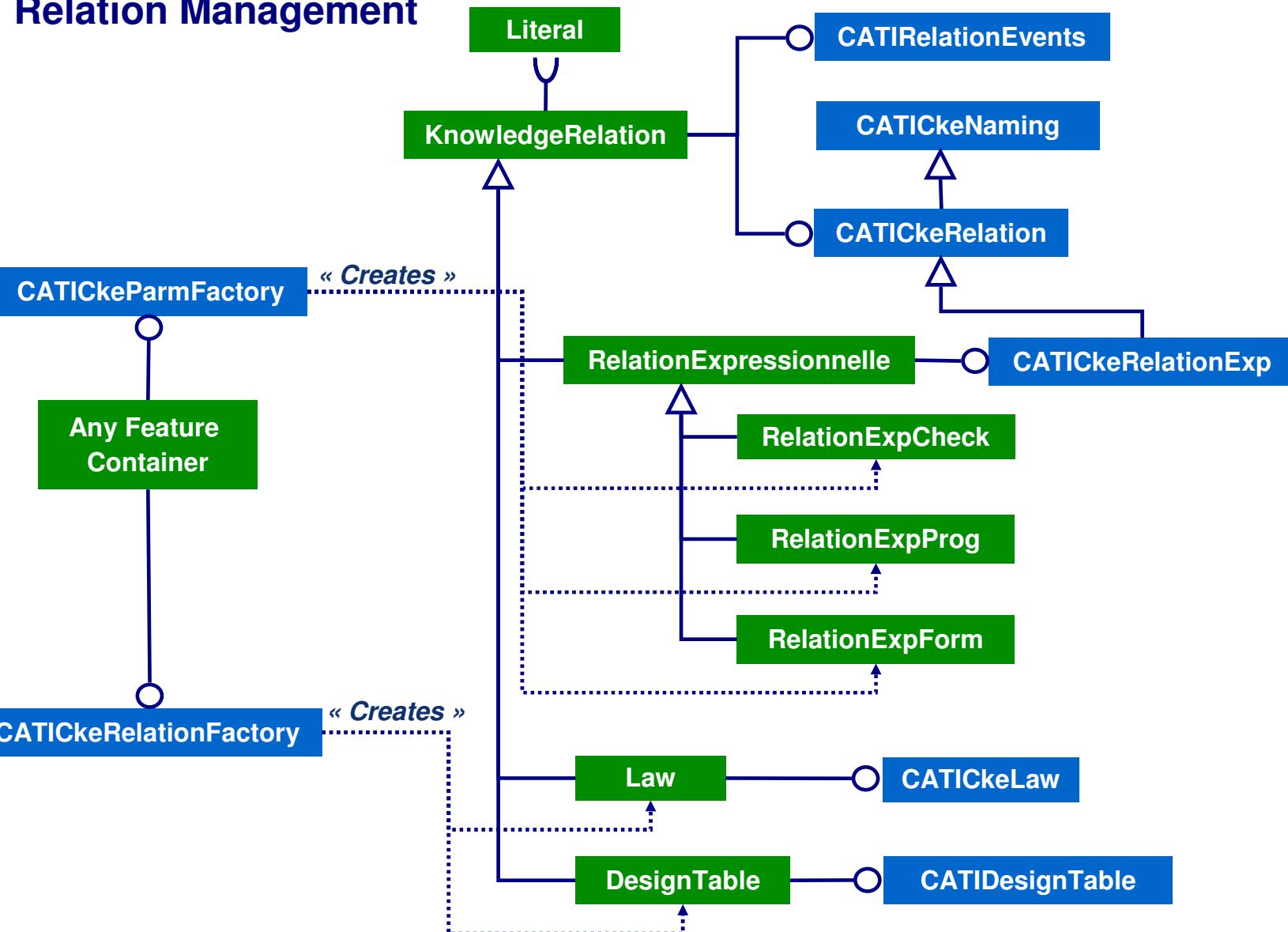
Relation

[Student Notes:](#)

- ◆ **Relation**
 - ◆ Defines a relation between one or several literals and how output values are deduced from input ones.
- ◆ **Design Table**
 - ◆ A relation type for which input is always an integer literal serving as an index in a table for valuating n other output literals.
- ◆ **Expressional Relation**
 - ◆ Relation between literals described by a language
 - ◆ Formula : evaluates one literal from other one(s)
 - ◆ Rule : evaluates several literals from other one(s)
 - ◆ Check : uses several literals as input and produces a diagnosis.

[Student Notes:](#)

Relation Management



[Student Notes:](#)

Relation creation sample (1/2)

Objective: Select a surface and compute its area through a formula

```
// Get the Param dictionary
CATIParmDictionary_var spParamDictionary = NULL_var;
spParamDictionary = CATCkeGlobalFunctions::GetParmDictionary();

// Get Param Factory
CATICkeParmFactory_var spParamFactory = spSpecCont;

// Creating object reference from selecting feature
CATICkeParm_var spiSurfaceRef = NULL_var;
spiSurfaceRef = spParamFactory ->CreateObjectReference(spiSurfaceAsSpec);

// Retrieving AREA magnitude
CATICkeMagnitude_var spiSurfaceMagnitude = NULL_var;
spiSurfaceMagnitude = spParamDictionary ->FindMagnitude("AREA");

// Creating & aggregating parameter
CATICkeParm_var spResultParam = spParamFactory ->CreateDimension(...)
```

Student Notes:

Relation creation sample (2/2)

```
// Creating formula '[F]Area'
CATCkeListOf(Parm) ListOfParm;
ListOfParm.Append (spiSurfaceRef );

CATICkeRelation_var spAreaAsRelation = NULL_var;
spAreaAsRelation = spParamFactory ->CreateFormula("[F]Area", //Name of the formula
                                                     "",           // blank
                                                     "",           // blank
                                                     spResultParam, // constrained parameter
                                                     &ListOfParm,   // Parameters used in form.
                                                     "area(a1)",    // formula expression where
                                                     ai corresponds to rank i object in parameter list.
                                                     NULL_var,      // NULL_var
                                                     CATCke::False); // False

// Get the Function factory
CATICkeFunctionFactory_var spFunctionFactory= NULL_var;
FunctionFactory= CATCkeGlobalFunctions::GetFunctionFactory();

// Init the Kware Libraries Methods
spFunctionFactory->Methods();

spFunctionFactory ->AddToCurrentRelationSet (spAreaAsRelation );
```

AddToCurrentRelationSet allows you to aggregate the formula, so keep it persistent and so it can be seen in the specification tree

Student Notes:

Parameter and Relation Set

Parameter Set

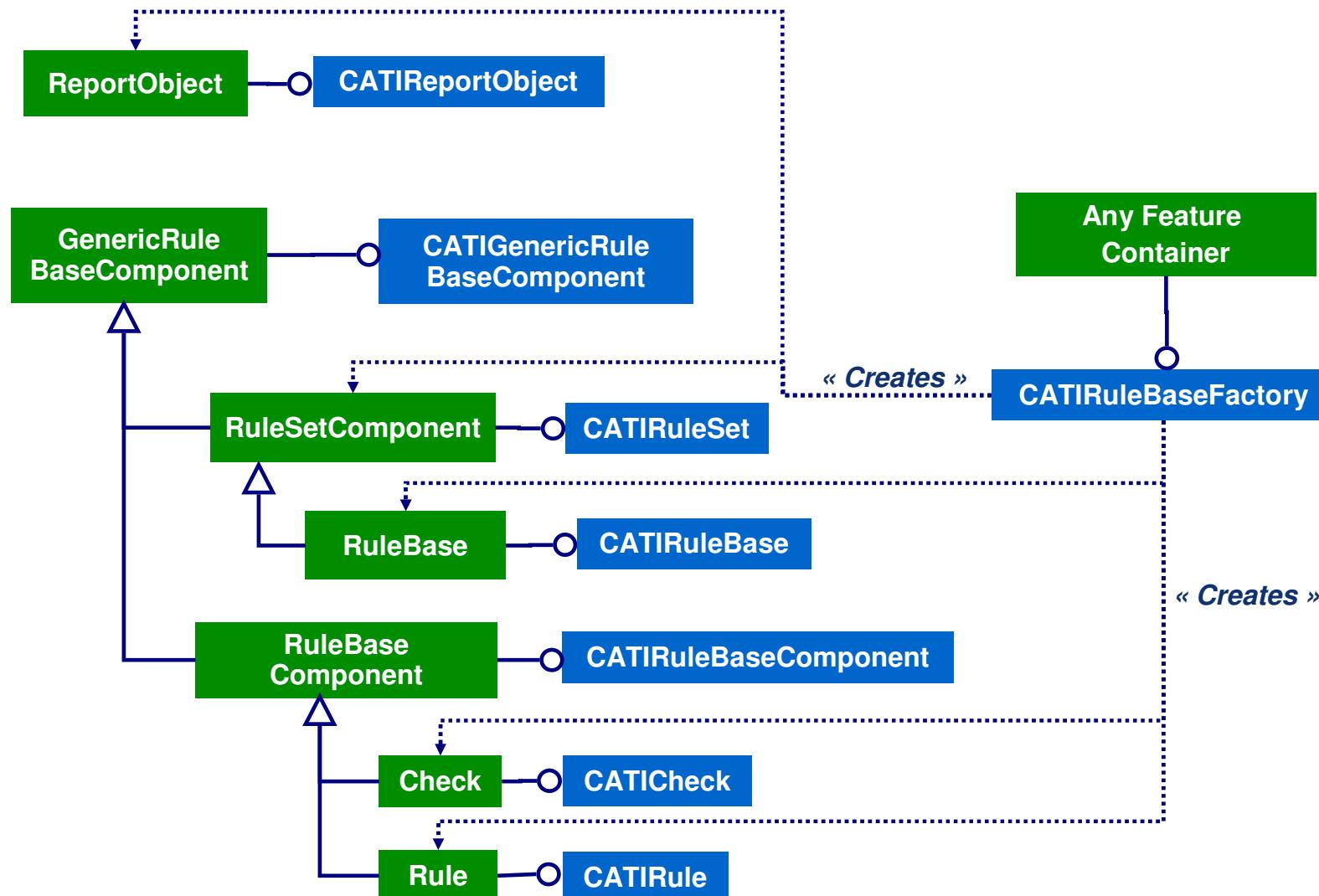
- ◆ Creates a Parameter Set viewed as a *Parameter Publisher*.
→ *CATICkeRelationFactory::CreateParameterSet*

Relation Set

- ◆ Creates a Relation Set viewed as a *Parameter Publisher*.
→ *CATICkeRelationFactory::CreateRelationSet*

Student Notes:

Knowledge Expert (*KnowHow*) Architecture



[Student Notes:](#)

Knowledge Expert Management [*KnowHow*] Interfaces (1/2)

- **CATICheck**
 - ◆ Defining the basic possibilities of an *Expert Check* which goal is to check a condition over a kind of objects
- **CATICheckReport**
 - ◆ Retrieving information on the check
- **CATIGenericRuleBaseComponent**
 - ◆ Defining the basic properties of the Knowledge Expert hierarchy of RuleBase, RuleSets, Checks and Rules.
- **CATIReportObject**
 - ◆ Dedicated to the Behavior of an Element of a Report.
- **CATIRule**
 - ◆ Describing Expert Rules
- **CATIRuleBase**
 - ◆ Dedicated to the RuleBase behavior.

[Student Notes:](#)

Knowledge Expert Management [*KnowHow*] Interfaces (2/2)

- ❖ **CATIRuleBaseComponent**
 - ◆ Describing the common contents of leaf nodes of the Knowledge Expert hierarchy.
- ❖ **CATIRuleBaseFactory**
 - ◆ Describe the Factory for Creation of RuleBases, RuleSets, Rules and Checks.
- ❖ **CATIRuleSet**
 - ◆ Dedicated to elementary operations on a Rule Set.

Exercise Presentation → Literal Volume Formula Creation in Interactive Mode.

[Student Notes:](#)

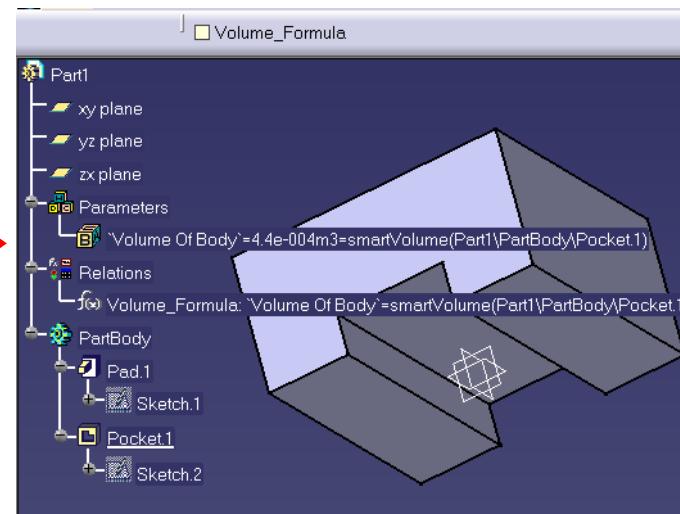
And now practice on the 5 steps of the exercise, to learn about:

- ◆ Initialization of Knowledgeware Tools
- ◆ Create an Object Reference from the CATBody and GetBack a Parameter
- ◆ Create a Volume type parameter
- ◆ Create a Formula which calculates the Volume
- ◆ Update the Relation for seeing Result

Initial State



Final State



[Student Notes:](#)

CAA V5 Product Structure & Assembly

You will learn what are the CATIA Product Structure and Assembly capabilities, and which frameworks are involved to provide them.

- Interactive Product Structure and Assembly Design Notions
- Objectives
- Programming aspects

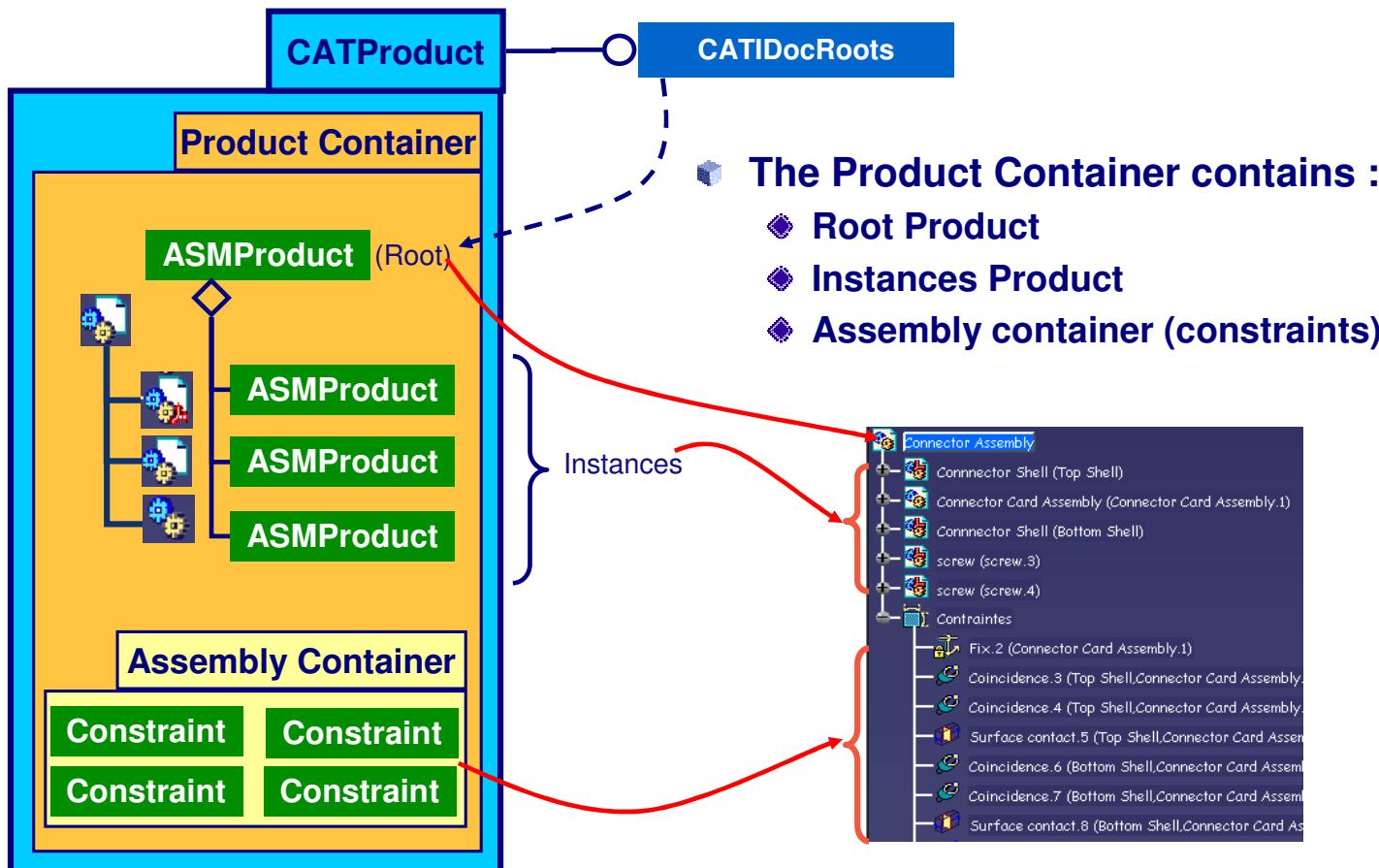
Student Notes:

CATIA Product Structure and Assembly Objectives

- Define the structure of a final product
 - ◆ A product is constituted by components.
 - ◆ These components can be a part (an atomic element) or another product (a sub-assembly).
 - ◆ It manages different types of document to represent a component.
 - ◆ Components can be set in position to each other.
- Provide the infrastructure of the CATProduct document
- Provide also the capability to integrate data linked to the Product Structure
- Provide the capability to create connections between objects belonging to different components.

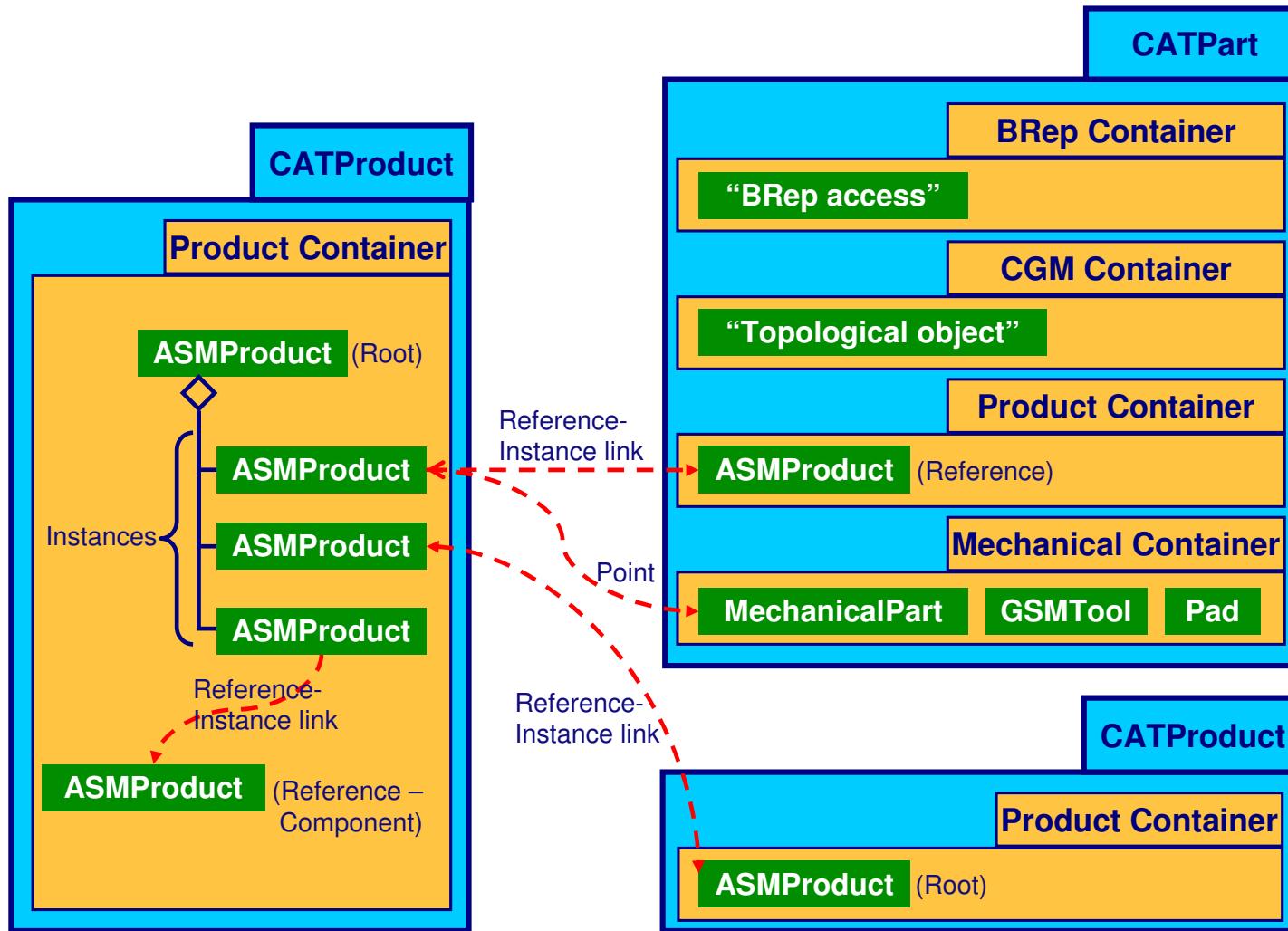
[Student Notes:](#)

The CATProduct Document overview



Student Notes:

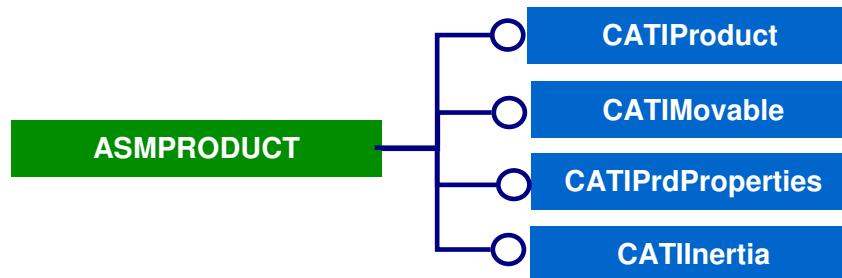
Product Instance and Reference Product



[Student Notes:](#)

Product Structure : ASMPRODUCT Feature

- Reference Products and Product Instances are ASMPRODUCT features
- CATIProduct manages instantiation of given references...
- CATIMovable manages the relative displacement with respect to a context (that can be another CATIMovable)
- CATIPrdProperties retrieve the product properties and CATIIInertia the mechanical properties of a product



Student Notes:

Product Structure : Root Feature

Retrieve the Root Product feature

```
CATDocument * pDoc = NULL;  
rc = CATDocumentServices::New("Product",pDoc);
```

```
CATIDocRoots * piDocRoots = NULL;  
rc = pDoc->QueryInterface(IID_CATIDocRoots, (void**) &piDocRoots);
```

```
CATLISTV(CATBaseUnknown_var) * pRootProducts =  
    piDocRoots->GiveDocRoots();  
CATIPruct_var spRootProduct = NULL_var;  
if (pRootProducts->Size())  
{  
    spRootProduct = (*pRootProducts)[1];  
    delete pRootProducts;  
    pRootProducts = NULL;  
}
```

The CATProduct document implements the CATIDocRoots interface.

From the CATIDocRoots interface, the ASMProduct features are retrieved: the root product + potential component products (those created through the “New Component” command).

The very first one corresponds to the Root Product.

[Student Notes:](#)

Product Structure : CATIProduct Interface

- ❖ Instantiate an existing Reference Product
 - ◆ Open the existing document
 - ◆ Retrieve the Root Product feature of this document
 - ◆ Use it as the argument of the AddProduct method of the CATIProduct interface
 - ◆ It returns a Product Instance

```
CATIProduct_var AddProduct( CATIProduct_var & iProduct,  
                           const CATIContainer_var & iCont=NULL_var)
```

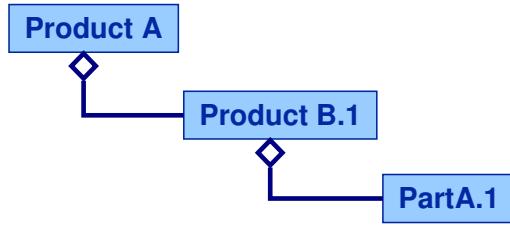
- ❖ Inserting a new component
 - ◆ Use directly the AddProduct method with a character string. It creates a component and adds it as sub-product
 - It creates also the Reference Product associated to the component and aggregates this Reference in the Product Container
 - ◆ It returns a Product Instance

```
CATIProduct_var AddProduct( const CATUnicodeString & iString,  
                           const CATIContainer_var & iCont = NULL_var)
```

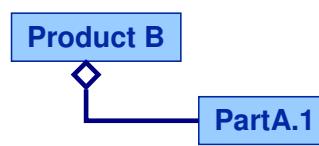
Student Notes:

Product Structure : Product Context

- When the product tree has more than 2 levels, some product instances will exist in different contexts
 - A context is associated to a root product



Product A context



Product B context

Part A.1 exists as Product instance in both contexts

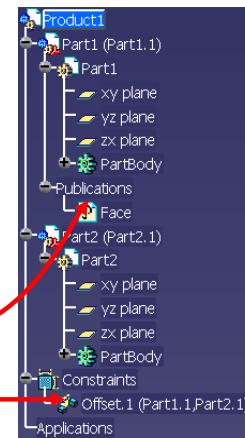
- To get a product instance into an other context use **CATIProduct::FindInstance()**
 - Provide as input the root product where to search for the instance

```
CATIProduct_var FindInstance( const CATIProduct_var & iProduct )
```

[Student Notes:](#)

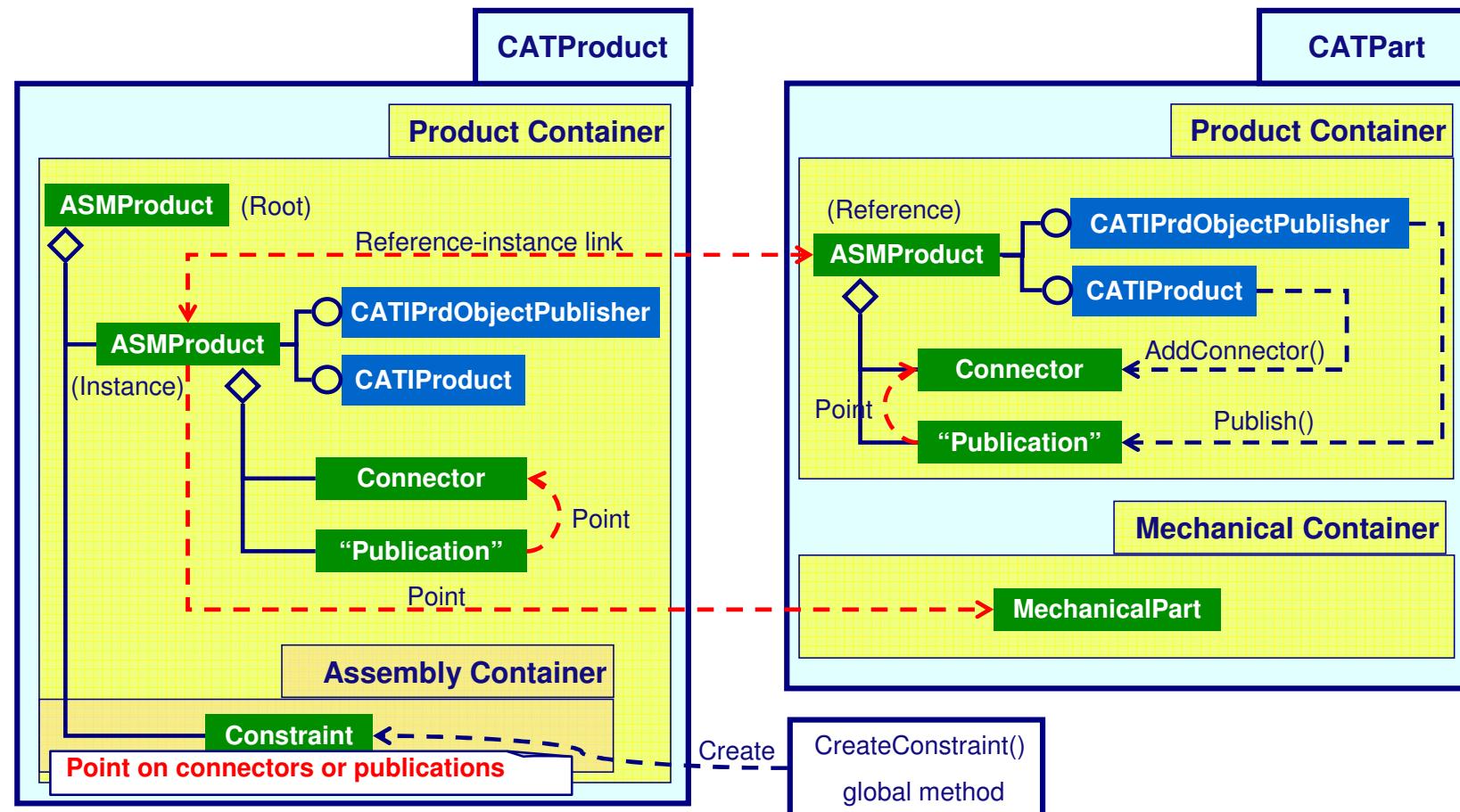
Product Structure : Connectors introduction

- ❖ A Connector insures a stable access to a topological object from an other container (Product, Analysis, Drafting, ...)
 - ◆ Storing the generic name
 - ◆ It can recreate a BRep access object dynamically
- ❖ Inside the assembly context,
 - ◆ It is built thanks to the `CATIProduct::AddConnector()` method which has as input a BRep Access
 - ◆ “Replace Component” supports Connectors
- ❖ From a connector, you can retrieve:
 - ◆ The corresponding BRep access object with the `GiveReferenceObject()` method
 - ◆ the position matrix with the `GiveContext()` method
- ❖ For instance, they are used to create:
 - ◆ Publications
 - ◆ Assembly constraints
 - ◆ ...



Student Notes:

Product Structure : Connectors architecture



[Student Notes:](#)

Product Structure : CATIConnector Use

```
....  
CATPathElement * Path = AcqAgent -> GetValue();  
CATILinkableObject_var spLinkable = AcqAgent -> GetElementValue();
```

Retrieve the CATILinkableObject interface of the selected object

```
CATIProduct_var spLeafProduct = Path ->FindElement(CATIProduct::ClassId());
```

Retrieve the Product instance the selected object belongs to

```
...  
  
CATIConnector_var spConnector;  
spLeafProduct->AddConnector(spLinkable, spConnector);
```

Create and aggregate a new Connector only if spLinkabke is not already referenced by an existing Connector

Student Notes:

Product Structure : CATIConnector Use (2/2)

■ Create a Constraint using connectors

```
....  
CATLISTV(CATBaseUnknown_var) ConnectorList;  
ConnectorList1.Append(spConnector1);  
ConnectorList1.Append(spConnector2);  
CATICst* piConstraint=NULL;  
  
HRESULT ResCstCreation=CreateConstraint( catCstTypeOn,  
                                         ConnectorList,  
                                         NULL,  
                                         spRootProductOfProduct,  
                                         &piConstraint);
```

Product Structure: Publications

Student Notes:

- The interface to manage Publications is **CATIPrdObjectPublisher**
- This interface allows you to :
 - ◆ Publish objects
 - ◆ Retrieve the object referenced by a publication
 - ◆ Retrieve all the publications of a given product
 - ◆ Retrieve all the publications that reference a given object

Product Structure: CATIPrdObjectPublisher Use

Student Notes:

```
....  
CATIConnector * ConnectorOnObjectToPublish = .....;  
....  
CATIPrduct_var myProduct = .....;  
....  
CATIPrdObjectPublisher_var myPublisher = myProduct;  
  
CATUnicodeString NameOfPublication= "Face";  
myPublisher -> Publish(&NameOfPublication, (CATBaseUnknown *) ConnectorOnObjectToPublish);
```

See previous slides

Retrieve the product that wants publish an object

Student Notes:

Exercise Presentation → Product Structure & Assembly Exercise

And now practice on two exercises, to learn about:

◆ **Scan a product structure**

- retrieve the children of a given product
- Apply a transformation to one of the part

◆ **Create a Product structure with constraints between two parts**

- Instantiate components from existing CATPart files
- Retrieve the component 's publications
- Create constraints between components using their publications

Student Notes:

CAA V5 Development Environment: Quality Control

You will learn about all the rules to follow to avoid basic coding errors

- Programming Rules
- C++ Source Checker: mkCheckSource
- Test Tool: mkodt
- Documentation Generation: mkMan

Student Notes:

CAA V5 Development Environment Quality Control Objectives

- Know the C++ programming rules used by Dassault Systemes. It helps to avoid naming conflicts of source files, Feature late types and to make the code more readable by other developers or Hotline team.
- Discover the wizard to detect as early as possible most common coding errors that lead to crashes at runtime.
- Put in place some basic tests of your application to detect before a new version's delivery some potential impacts of some code modifications. That guarantees data stability.

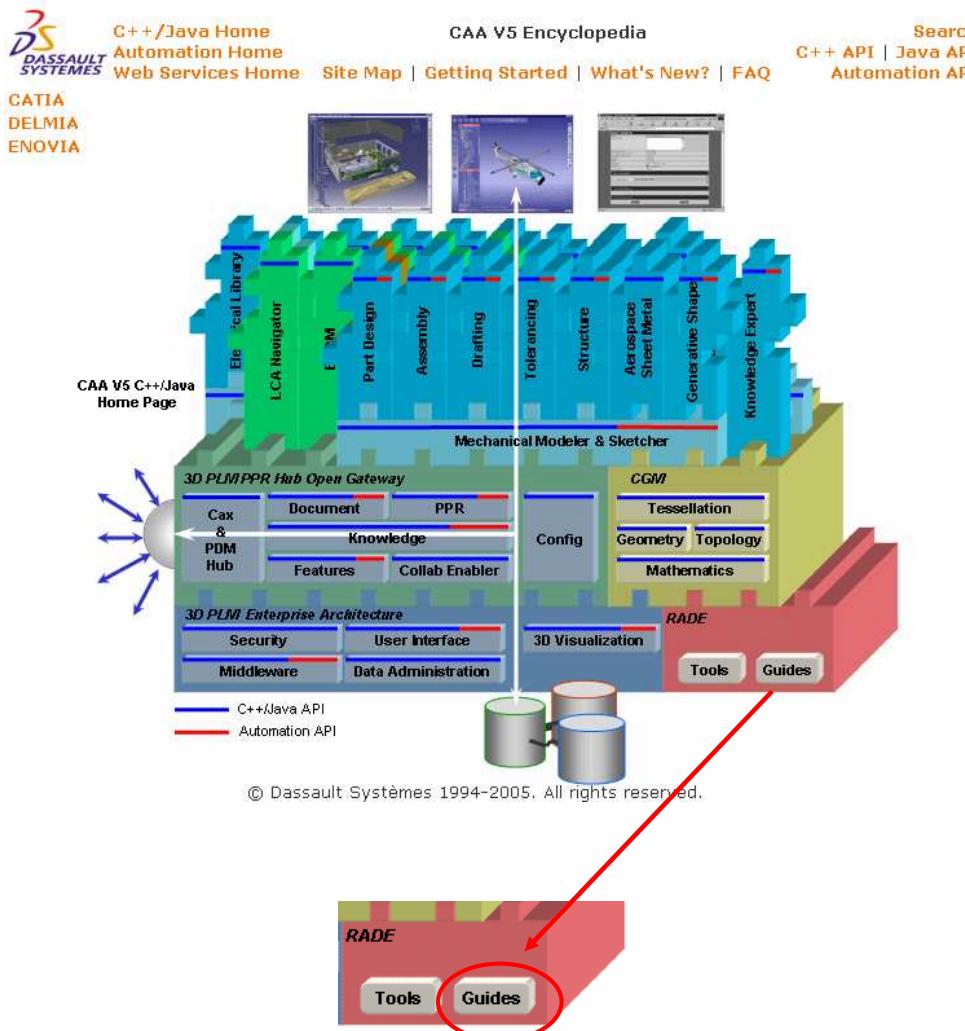
Student Notes:

CAA V5 C++ Programming Rules

That includes:

- Naming convention
- C++ Naming rules
- C++ coding rules
- Java coding rules
- Architecture rules
- User Interface look and feel

- Available in the encyclopedia



[Student Notes:](#)

CAA V5 Naming Convention

- **Naming conventions**
 - ◆ To avoid name collisions between the entities that can be found together in the same repository at run time.
 - ◆ Names are constituted by English names, each one starting with an uppercase.
- **Three letters alias for product name.**
 - ◆ CAT / VPM / ENOV / DNB reserved for DS product lines
 - ◆ Partner Name : DCS, MDI, CSC, ...
- **Three letters alias for each framework (here « MId »).**
 - ◆ **TSTMIdDesignFeature** (framework)
 - ◆ **TSTMIdComponent.m** (module)
 - ◆ **TSTMIdEjectorImpl.cpp/.h** (class)

Student Notes:

General rules

- Variable names begin with a lowercase letter (int counter;)
- Function names begin with an Uppercase letter (void CountItems();)
- * and & are to be collated to the variable name, not the type name, in order to avoid ambiguities in case of multiple definitions on the same line (V5Object *pFirstItem, *pSecondItem;)
- Data members are to begin with an underscore (int _length). Data members should never be assigned public (in the C++ sense) visibility

[Student Notes:](#)

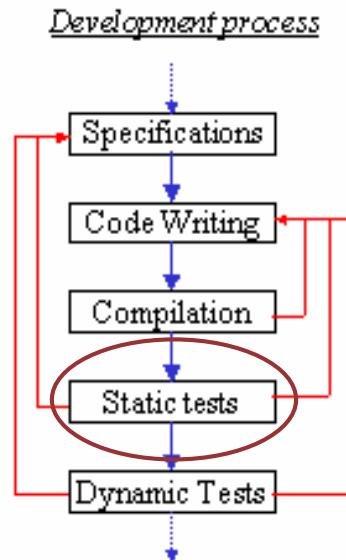
Lifecycle Rules

- Always initialize your pointers to NULL.
- Always test pointer values before using them.
- Always set pointers to deleted objects to NULL.
- Always set released interface pointers to NULL.
- Manage the Lifecycle of interface pointers
 - ◆ As a general rule, any interface pointer must be:
 - AddRef as soon as it is copied
 - Released as soon as it is not needed any longer.
 - ◆ A call to Release must be associated with each call to AddRef
- Manage the Lifecycle of objects that are not interface pointers
 - ◆ As a general rule, associate a delete with each new, or a free with each malloc.

Student Notes:

C++ Source Checker: mkCheckSource

- mkCheckSource (C++ Source Checker) is a C++ static controller which analyzes a program and tries to discover potential defects or logical errors without executing the program.

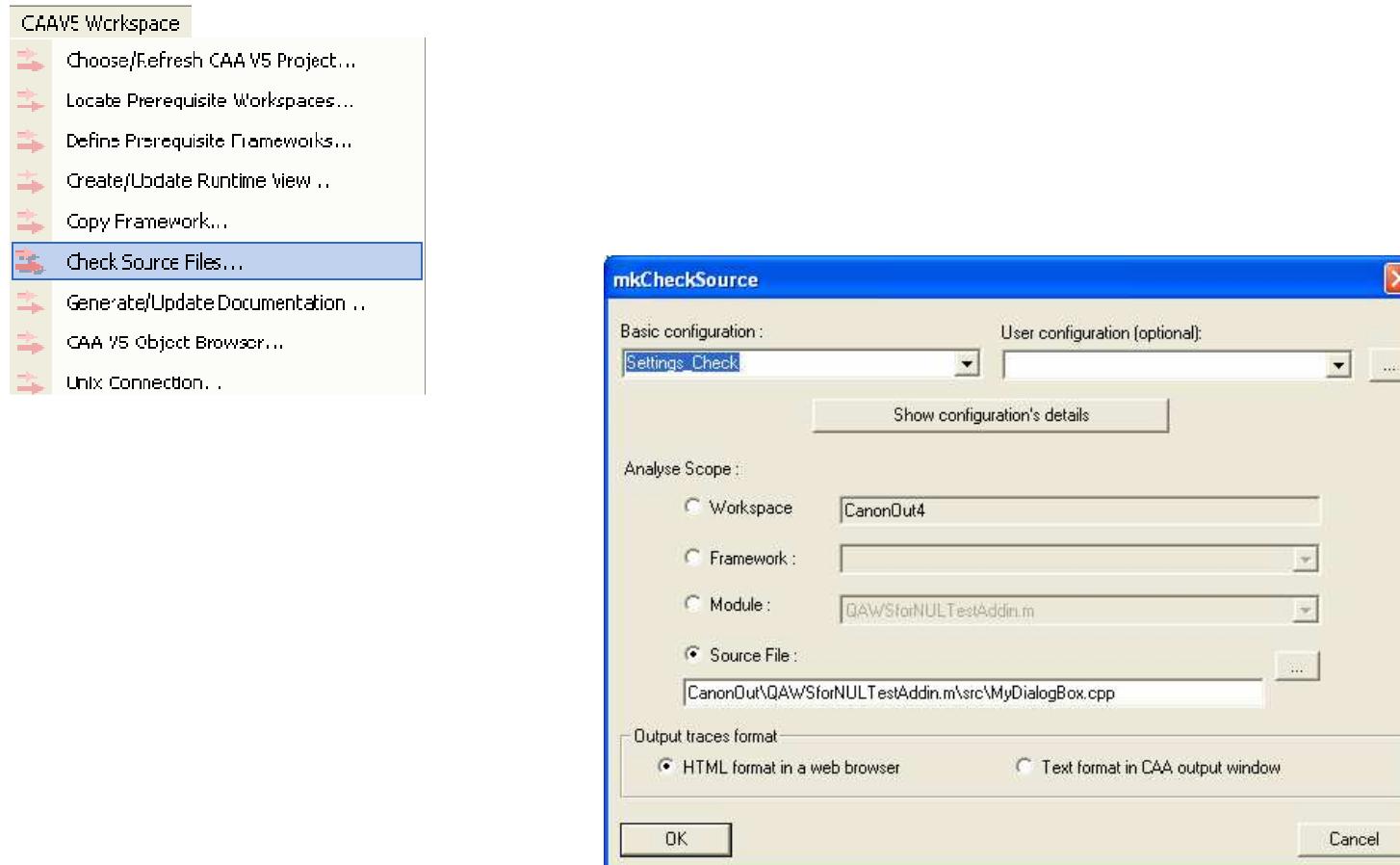


- Available in the encyclopedia



Student Notes:

mkCheckSource CAA wizard



[Student Notes:](#)

Other Tools used in the CAA V5 context

Rational Purify

- ◆ A tool to detect any memory leak and to be used with mkodt
- ◆ An automatic debugger allows to detect predefined errors during a program compilation or by checking his run.
- ◆ For example, classical detected errors are writing or reading a non allocated block or reading an uninitialized memory block.
- ◆ Use : mkodt –s MyTest.sh -B purify –b cnext

Rational Pure Coverage

- ◆ A tool to check the percentage of the code really tested and to be used with mkodt
- ◆ Use : mkodt –s MyTest.sh -B purcov –b cnext

[Student Notes:](#)

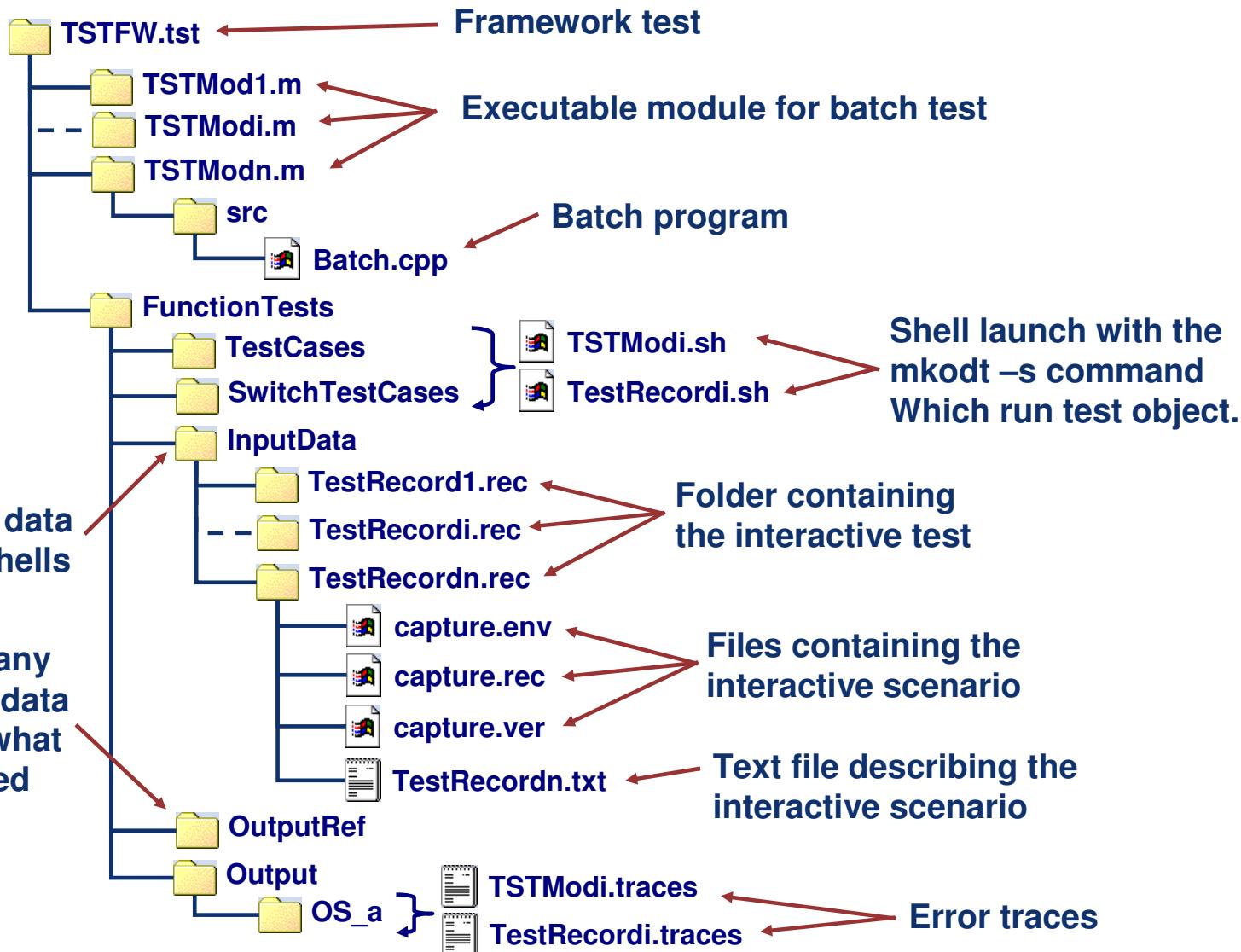
Test Tool: mkodt

- Like for development tasks, tests activities can take place in your environment.
- RADE provides an infrastructure for writing, delivering, and running test programs associated with CAA V5 frameworks.
- There are two kind of test object:
 - ◆ Batch Test
 - ◆ Record Test (interactive way)
- These test object are created in a special framework (with .tst extension) and launch thanks to a shell.



Student Notes:

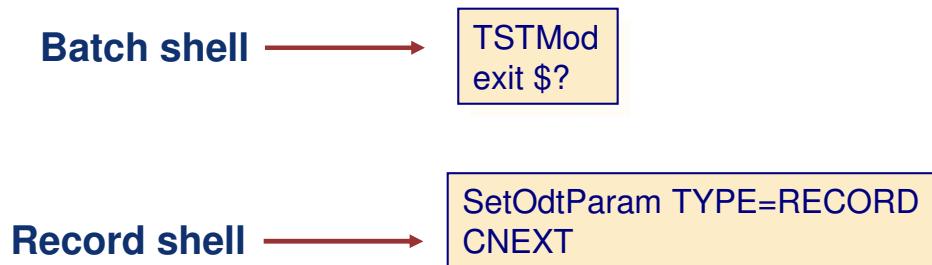
Test Framework Architecture



Student Notes:

Shell: Basic Sample

- Shell variables are automatically set according to the current test environment
- Basic Shell sample:



- As you can see, the minimum consists of first running the program and secondly exiting with the program return code.

Shell: Example

Student Notes:

- Value variables



```
#-----
export TEST=YES
export DICO=$ADL_ODT_IN
#-----
my_program <$ADL_ODT_IN/in1_mod1
rc=$?
if [ $rc != 0 ]
then
  exit $rc
fi
diff $ADL_ODT_OUT/out1_mod1 $ADL_ODT_REF/out1_mod1
rc=$?
#-----
exit $rc
```

- Set input data



- Compare outputs with a reference



[Student Notes:](#)

Shell: Environment Variables

- **ADL_ODT_TMP**
 - ◆ (Read only) References a temporary directory, for instance /tmp on Unix and C:\TEMP on Windows.
- **ADL_ODT_IN**
 - ◆ (Read only) References a directory where all data needed as input for a program are stored.
- **ADL_ODT_OUT**
 - ◆ (Read/write) References an output directory where any output data can be stored
- **ADL_ODT_REF**
 - ◆ (Read only) References a directory where reference data are stored. Reference data are data used to compare what a program produces with what should be produced....
- **ADL_ODT_NULL**
 - ◆ (Read only) References a special file which can be used to redirect outputs from a program (/dev/null with Unix or NUL with Windows.)
- **ADL_ODT_CONCATENATION**
 - ◆ (Read/write) References a concatenation

[Student Notes:](#)

Shell: SetOdtParam Instruction

- A shell can include the SetOdtParam instruction to control some aspects of its execution
- **SetOdtParam max_time=xx**
 - ◆ max_time is the maximum time allowed for test to be completed. The default value of max_time is 5 minutes.
- **SetOdtParam Replay_xxxx=NO**
 - ◆ xxxx can be AIX, IRIX, SunOS, HP-UX, Unix, Windows_NT, aix_a, irix_a, solaris_a, hpx_a, intel_a.
- **SetOdtParam USER=xxx**
 - ◆ This instruction is used to specify the user (xxx) when the ODT will be replayed.
- **SetOdtParam ADL_ODT_XX=xxx**
 - ◆ To evaluate environment variables

[Student Notes:](#)

Shell: script rules

Forbidden

#!/bin/ksh
No export of PATH, LIBPATH and ADL_ODT_xxx
Don't use the rm and mv commands
Don't use "/" in the PATH
Don't use ":" in the PATH
Don't run process in background (process&)
No file Path: /myUser/...
Don't use /tmp
Don't use /dev/null
-
return xx
..../InputData (relative Path)
/dir/WS/OS/code/bin/LOAD
No double bracket [[...]]

Authorized

-
Use \$ADL_ODT_SLASH
Use \$ADL_ODT_SEPARATOR
-
Use ADL_ODT_xxx variables
Use ADL_ODT_TMP
Use ADL_ODT_NULL
Test the return code
exit xx
Use ADL_ODT_xxx variables
LOAD
Use [...]

Student Notes:

Interactive scenario

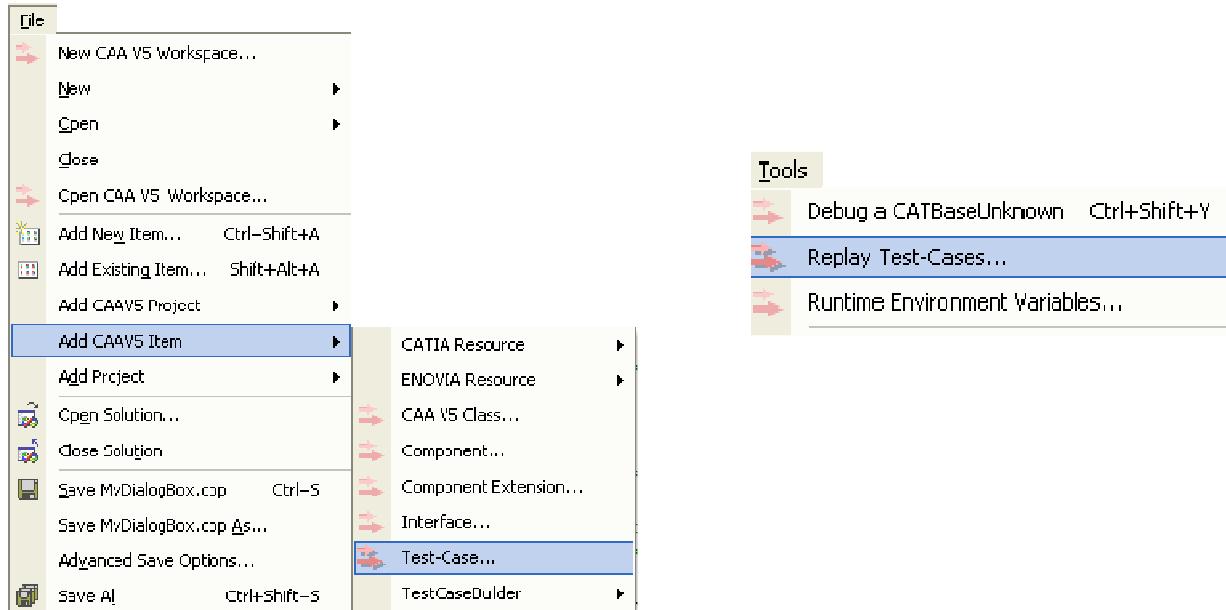
- Interactive scenario is stored inside a folder called FW.tst\Inputdata\xxx.rec.
- Inside this folder, you can create a text file to explain your interactive scenario (it could be useful if you want to store again this scenario).
- Use the mkodt -s RecordTest1.sh –C command to launch the capture scenario.
- CATIA is launched and you can record your interactive scenario.
- To finish your scenario, you have to click on file close and file exit (do not save your document and do not use the cross icon).
- To Replay your scenario use mkodt -s RecordTest1.sh
- Use –X option to see the traces : mkodt -s RecordTest1.sh –X

[Student Notes:](#)

mkodt command

- Run a Test: **mkodt** command
- **mkodt options:**
 - ◆ Available by **mkodt -h**
- Mapping between commands and MsDev Add-Ins:

mkodt



[Student Notes:](#)

mkodt options

- **-a**
 - ◆ Replays all tests objects for a workspace, a framework, or a directory TestCases or SwitchTestCases
- **-F framework**
 - ◆ Specifies a framework directory or a list of framework directories of the workspace
- **-k**
 - ◆ Keeps the temporary directory set using ADL_ODT_TMP
- **-NOMaxTime**
 - ◆ Disables the supervision of the ODT's execution time
- **-o output**
 - ◆ Specifies an output directory and sets the ADL_ODT_OUT variable. Results of mkodt (.traces and .mkodt_results files) are generated in the directory Output/Framework.tst/FunctionTests/Output/OSName
- **-s shell**
 - ◆ Lists the shell script names to run

[Student Notes:](#)

mkodt options

■ -B debugger

- ◆ Turns the debugger mode on and specifies the debugger to use. Useful with the -b option. According to the operating system, available debuggers are (first one is the default):
 - Solaris workshop, dbx
 - Irix cvd, dbx
 - AIX xlDb, dbx
 - HP_UX dde
 - Windows msdev, purify, purcov

■ -b program

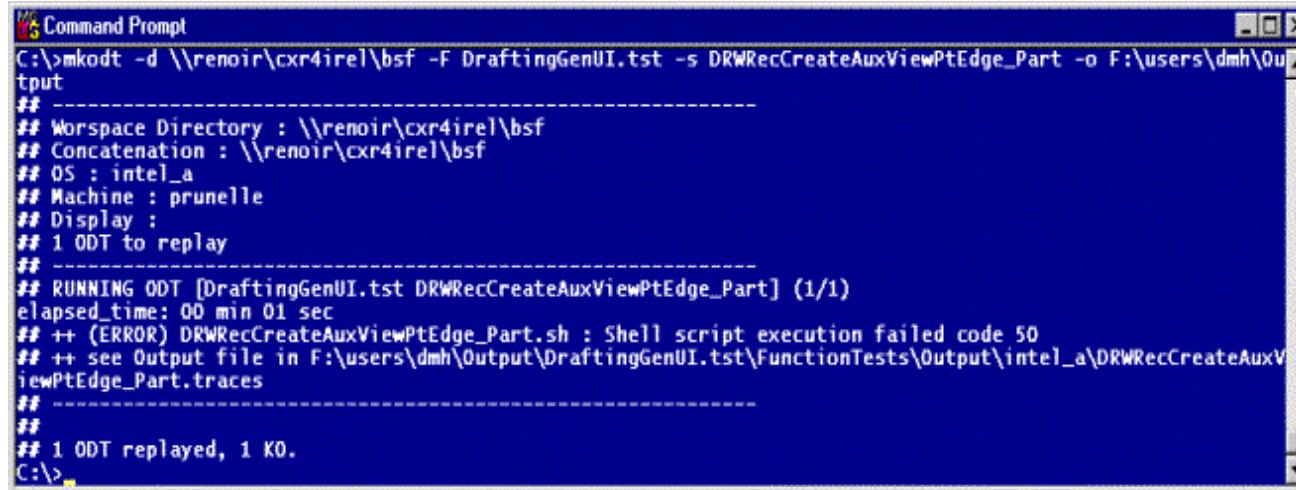
- ◆ Uses the debugger on the given program. It must be a module or an EXE containing a main.

Example : mkodt –B msdev –b CNEXT –s MyOdtShelScript.sh

[Student Notes:](#)

mkodt outputs

Traces window:



The screenshot shows a Windows Command Prompt window titled "Command Prompt". The command entered is "C:\>mkodt -d \\renoir\cxr4irel\bsf -F DraftingGenUI.tst -s DRWRecCreateAuxViewPtEdge_Part -o F:\users\dmh\Output". The output shows mkodt configuration parameters like workspace directory, concatenation, OS, machine, display, and ODT replay. It then indicates it is running the ODT [DraftingGenUI.tst DRWRecCreateAuxViewPtEdge_Part] (1/1) and shows an error message: "Shell script execution failed code 50" with a trace file path "F:\users\dmh\Output\DraftingGenUI.tst\FunctionTests\Output\intel_a\DRWRecCreateAuxViewPtEdge_Part.traces". Finally, it shows "1 ODT replayed, 1 KO." and ends with "C:\>".

Output files content:

- ◆ The mkodt command generates two output files in the Output/\$OS directory.
 - xxx.traces: contains traces generated by the shell script.
 - xxx.mkodt_result: contains result of the ODT replay.

[Demo Video](#)

[Student Notes:](#)

Documentation Generation: mkmancpp

- Interfaces, classes, and all the stuff they include, are documented thanks to information enclosed into specific comments inserted in the header files.

```
/**  
 *This is a comment.  
 */
```

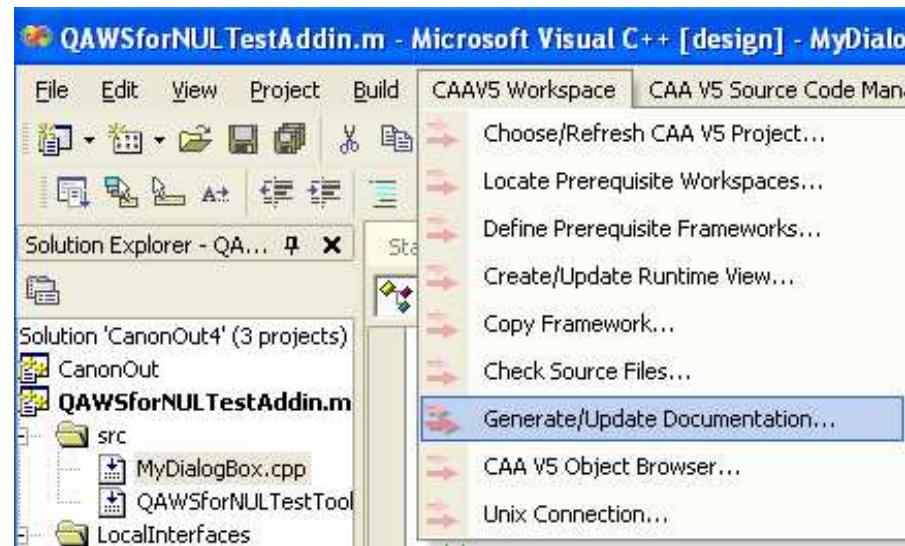
- At the top of the header file you have to put the following tags :

```
/**  
 * @CAA2Level L1  
 * @CAA2Usage U1  
 */
```

- The C++ statements and the comments are then processed to produce an organized set of html files in which you and the users of your interfaces and classes can navigate.
- mkmancpp is the command that allows to generate such a documentation.

MsDev Add-In

[Student Notes:](#)



Exercise Presentation → Development Environment Quality Control

And now practice two exercises, to learn about:

- ◆ Run mkmancpp on the TSTScrewExercise
- ◆ Create a TestCase (shell) in a test Framework
 - Create the Test Framework
 - Create a Batch Module
 - Create a Shell
 - Run mkodt command

[Student Notes:](#)

Student Notes:

CAA V5 Administration

You will learn the CAA packaging, the prerequisites in order to use CAA and how to deliver the applications built with CAA

- Packaging and Licensing
- Software Prerequisites
- Delivering a CAA built Application

CAA V5 Physical Packaging

Student Notes:

- **Rapid Application Development Environment (one CD)**
 - ◆ A set of programming tools to be plugged in standard interactive development tools
- **V5 API for CATIA (one CD)**
 - ◆ A set of headers with their corresponding documentation
 - ◆ The CAA V5 Encyclopedia
- **Program directory (one CD) :** Don't forget to check this bullets before the CAA installation
 - ◆ Hardware compliance
 - ◆ Software prerequisite
 - ◆ Installation procedure

Student Notes:

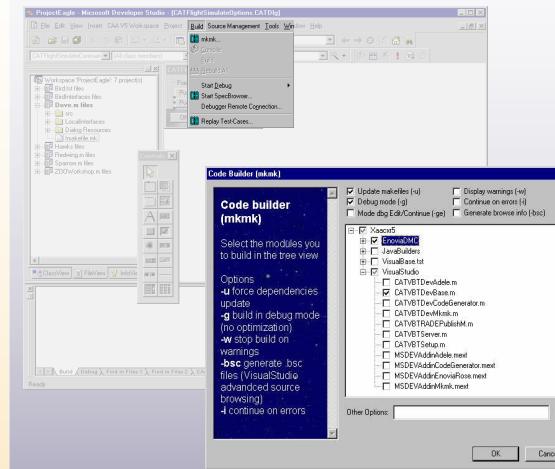
CAA V5 Installation Procedure

- **Check the CATIA installation**
 - ◆ You need to have the same Release and Service Pack as for CAA
 - ◆ The directory path for CATIA **mustn't contains blank characters**
- **Install CAA V5 API for CATIA + RADE**
 - ◆ The V5 API are installed in the same directory as CATIA
- **Install CAA V5 RADE**
 - ◆ Choose a separate directory to install the CAA V5 RADE product, the default one for instance
- **Work with the same level of CATIA, CAA API and CAA RADE.**
- **Installing CAA Service Pack update automatically the RADE service pack.**
- **The Service Pack Management applies on CAA API and CAA RADE in the same way as CATIA:**
 - ◆ **CAA API from CATIA Software Management Tool and the CAA API specific tab page**
 - Start + Programs + Catia + Tools + Software Management
 - ◆ **CAA RADE from RADE Software Management Tool**
 - Start + Programs + RADE + Tools + Software Management

Student Notes:

CAA –Multi-Workspace Application Builder (MAB)

- Multiple workspace compilation, link and run time creation
- Access to multiple compilers inc. C++, JAVA, IDL
- UNIX/Windows same capabilities
- ActiveX and CAB files support



DESIGN and IMPLEMENT

➤ **Consistent environment for multiple compilers**

➤ **Productive – build only what has been changed**

Student Notes:

CAA – C++ Interactive Dashboard (CID)

- Extends MS Visual C++ and .NET with specific V5 components through add-ins
- Portage from Windows to UNIX
- Interactive help on V5 source code
- For CATIA:
 - workbench and command creation wizard
 - multi-platform interactive panel builder
- For ENOVIA:
 - events customization wizard
 - Interactive user exit



DESIGN and IMPLEMENT

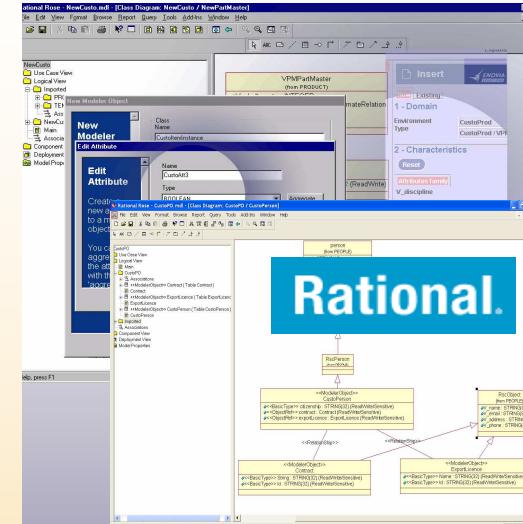
- Integrated in state-of-art standard tools
- Speeds development time
- Reduce learning cost
- Push button code generation

Student Notes:

CAA – Data Model Customizer (DMC)

- Extends Rational Rose with V5 components
- CATIA data features
- CATIA dialog engine
- ENOVIA modelers
- ENOVIA publication capability
- ENOVIA deployment on production site

DESIGN and IMPLEMENT

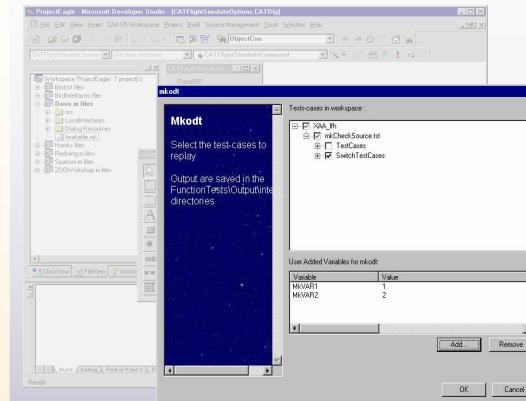


- Visual & interactive design of data model
- Integrated in state-of-art modeling tools (UML)
- A single tool for CATIA, ENOVIA & DELMIA V5

Student Notes:

CAA – C++ Unit Test Manager (CUT)

- ❖ Creation of test objects and automatic run time batch replay
- ❖ Memory management and runtime error checking:
 - ◆ Rational purify required
 - ◆ Windows only
- ❖ Enable C++ test coverage computation
 - ◆ Rational coverage required
 - ◆ Windows only



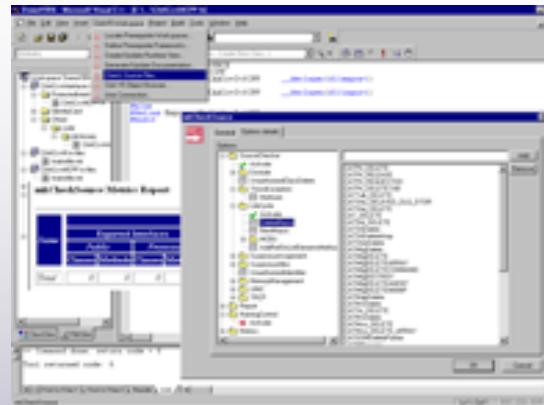
QUALITY TESTING

- Improves quality
- Enhance standard capabilities with additional products
- Better visibility on untested code

[Student Notes:](#)

CAA – Interactive Test Capture (ITC)

- Editing capabilities including debugging
- Multi-platform support for capture & replay
- No preemptive usage needed

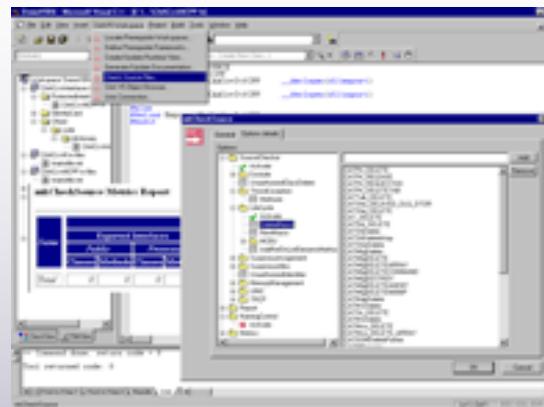


- Fast interactive user scenario recorder
- Cost effective solution
- Leverage C++ Unit Test Manager

[Student Notes:](#)

CAA – C++ Source Checker (CSC)

- Automatic check of C++ V5 coding rules
- Potential memory leaks identification
- Full HTML report



- Improves quality
- Reduces time spent for debugging
- Direct url access to faulty source line

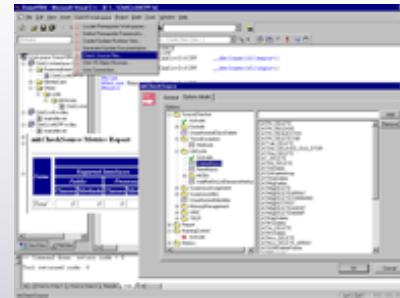
QUALITY TESTING

[Student Notes:](#)

CAA – C++ API Documentation Generator (CDG)

- ❖ C++ API documentation
- ❖ Pseudo-language to define the documentation sections
- ❖ Generated documentation includes:
 - ◆ Index file
 - ◆ Interfaces & class documentation attached to their framework
 - ◆ Framework list

INTEGRATION and DEPLOYMENT



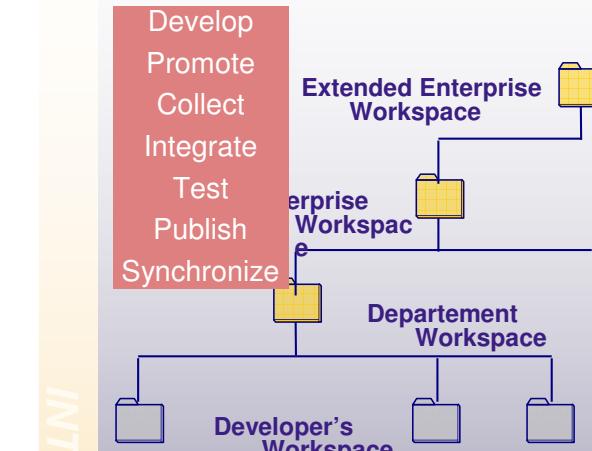
↗ **WEB centric documentation**

↗ **Allows customers to publish their API documentation**

Student Notes:

CAA – Source Code Manager (SCM)

- ❖ Configuration control
- ❖ Code merging
- ❖ Collaborative and distributed across:
 - sites
 - O.S. platforms
- ❖ Workspace management
- ❖ Hierarchical project integration



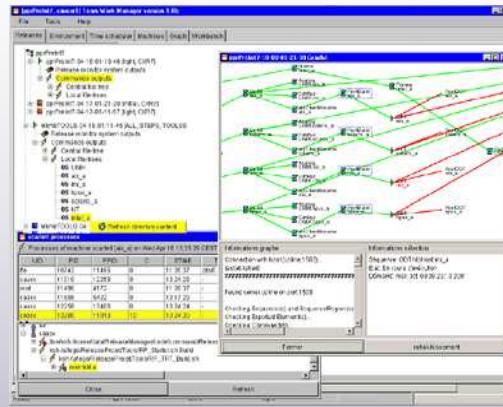
INTEGRATION and DEPLOYMENT

- Enables concurrent application development
- Software integration support
- Scalable
- Reliability and

Student Notes:

CAA – Teamwork Release Manager (TRM)

- Automatic and manual release management
- Launch of synchronous tasks over a multiOS network
- Single front-end interface
- Release tasks reporting features
- Schedules release operations through calendar function

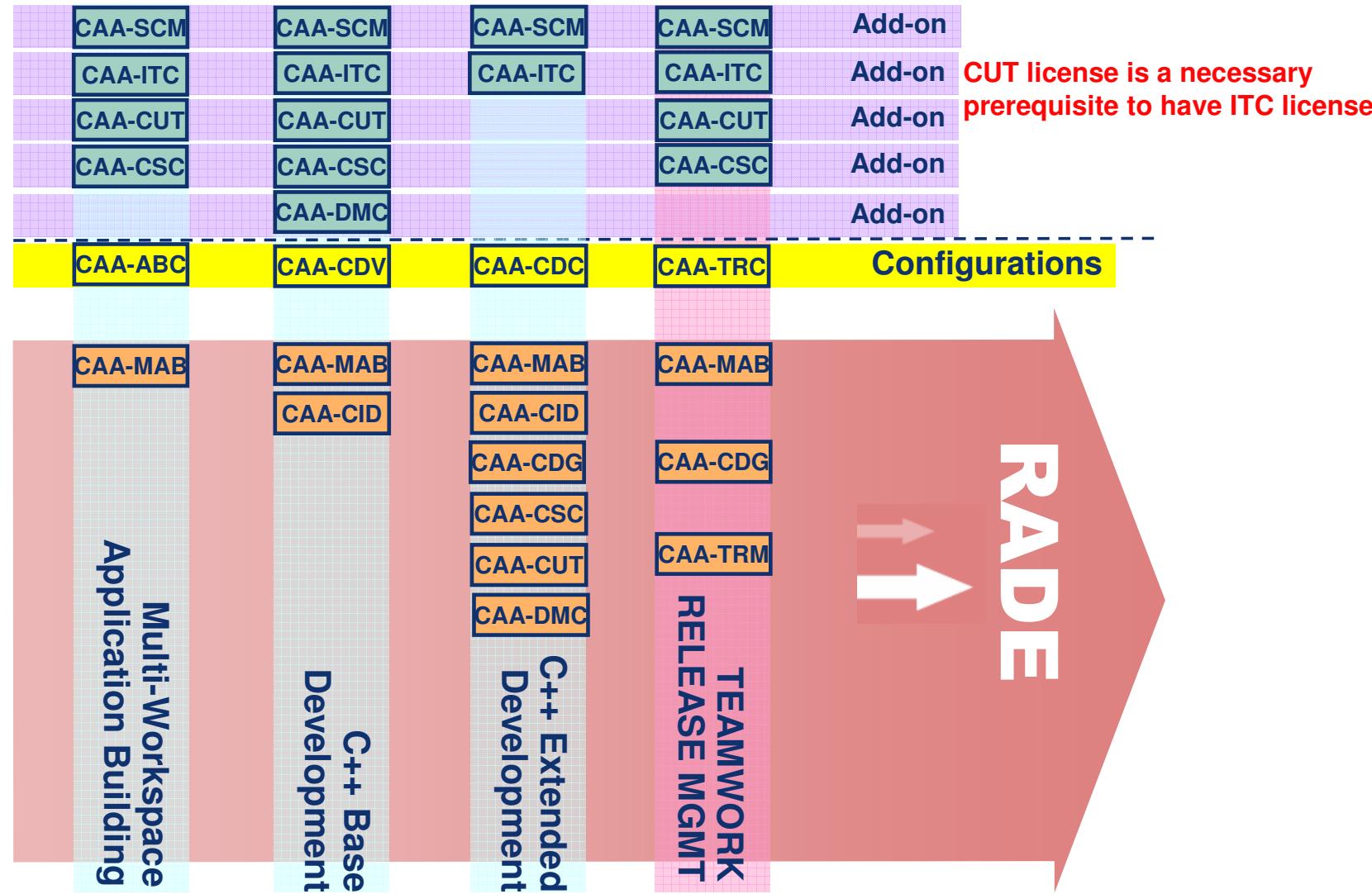


INTEGRATION and DEPLOYMENT

- Reduces release time cycle
- Ease-of-use
- Maximizing resource distribution

Student Notes:

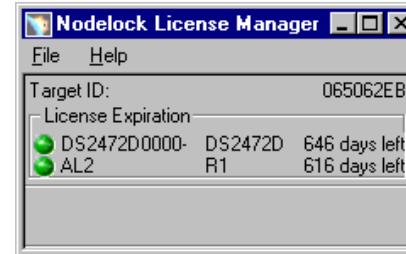
CAA V5 Packaging and Portfolio



Student Notes:

CAA V5 Licensing Mechanisms

- CDC or CDV or TRC license is required to use CAA V5 RADE (C++)
- CAA V5 can be used in two licensing modes just like standard DS products
 - ◆ Nodelock
 - ◆ Concurrent usage of licenses on a network.
- Nodelock Key Management
 - ◆ On WINDOWS, Select the Start->Programs->CATIA->Tools->Nodelock Key Management command to check Nodelock licenses
 - ◆ On UNIX, run CATNodelockMgt.exe

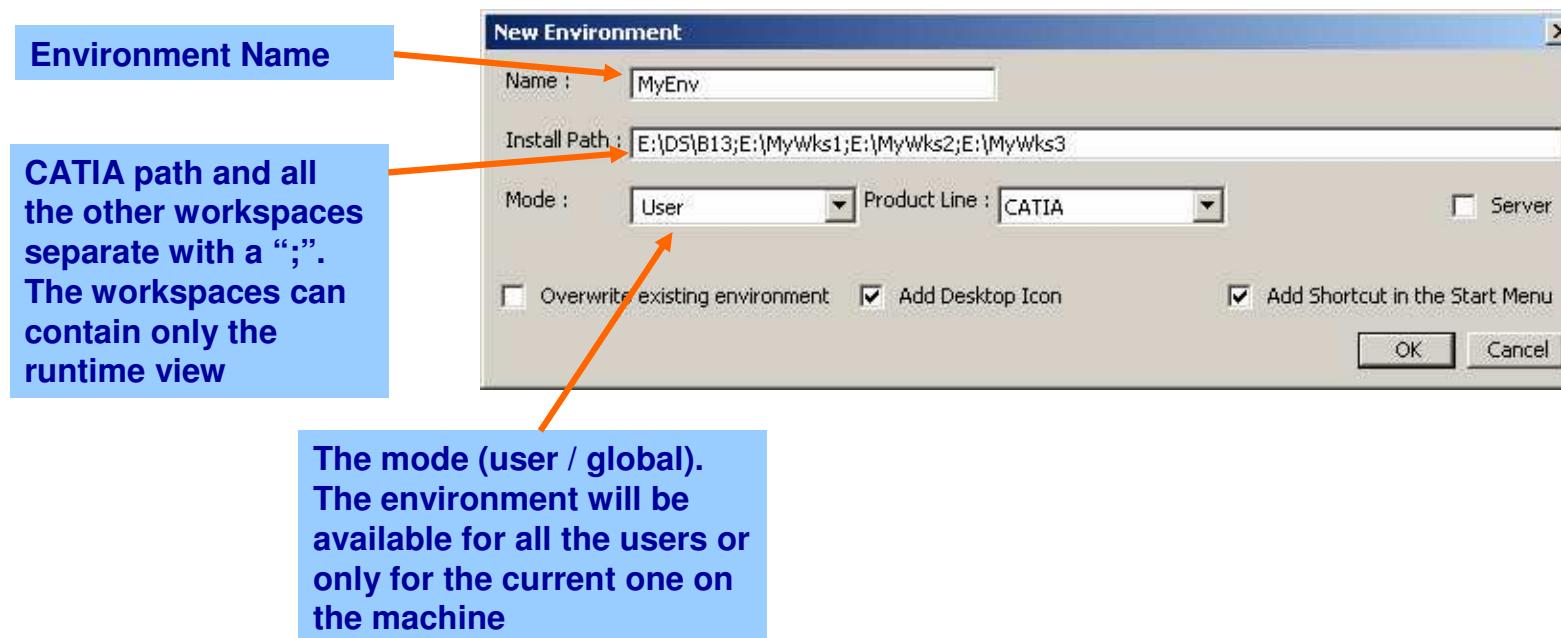


- Licensing mechanisms are based on LUM (IBM License Use Management)
 - ◆ Windows workstations must have a LAN Card (Ethernet or Token Ring) and TCP/IP installed and properly configured, even in the case of nodelock licensing

[Student Notes:](#)

Delivering an Application On Windows (with the environment editor)

- You can use the Environment Editor to create a new environment:
 - Select Start + Programs + CATIA + Tools + Environment Editor
 - Then Environment / New and the following panel appear:



- An icon (called MyEnv) will be created on your Windows Desktop which allow you to launch CATIA with the others add-in.

[Student Notes:](#)

Delivering an Application On Unix (with the environment editor)

◆ You can also use the environment editor

◆ Launch CATIAENV as following:

- .. / B14 / OS / code / command / catstart –run CATIAENV

◆ The Environment Editor panel will appear (like on Windows).

◆ In the .. / CATEnv folder you will find two files:

- MyEnv.sh : shell to set the new CATIA environment
- MyEnv.txt : text file which describe the environment

◆ To launch CATIA with the new environment, you can launch the following command :

- .. / B14 / OS / code / command / catstart –env myEnv –direnv myDirEnv

Student Notes:

Delivering an Application (without the environment editor)

- Use the ...\\B14\\OS_a\\code\\bin\\setcatenv command to create a new environment
 - ◆ The main options are :
 - -e : to specify the name of the new environment
 - -d: to specify the directory where the CATIA environments are stored
 - -p : to give the path
 - ◆ The path has to reference CATIA and the Runtime View of your CAA application
 - On Windows: setcatenv -e myEnv -d c:\\catia_v5 \\ CATEnv -p c:\\catia_v5 ; c:\\myApplication1; c:\\myApplication2 ; ...
 - On Unix: .. / B14 / OS / code / command / catstart –run “setcatenv -e myEnv -d / u / user / CATEnv -p / u / catia_v5 : / u / user / myApplication1 : / u / user / myApplication2 : ...”
 - If the path contains a blank character, use double quotes ("")
- To delete an environment use delcatenv
 - ◆ delcatenv -e myEnv
- To launch your application with a new environment
 - ◆ catstart –env myEnv –direnv myDirEnv to launch CATIA
 - ◆ Use –run MyApplication to launch your own executable.

[Student Notes:](#)

Runtime compatibility

- Since V5R9 Dassault Systèmes insure le RunTime compatibility of client application.
 - ◆ Client do not need to recompile their application at each service pack
 - ◆ Compatibility between Release are not insure, client MUST recompile the application for each Release
- The L1 tag
 - ◆ Every Dassault Systèmes header file (provided with CAA) has some specific tags that define it's quality.
 - ◆ You can find more information about Dassault Systèmes tags in the Encyclopedia->Rade->Guides->CAA V5 Resources Usage
 - ◆ You are supposed to use ONLY the L1 headers
- Runtime compatibility and L1 tag
 - ◆ The Runtime compatibility mechanism is based on the fact that you MUST use ONLY L1 headers. If you don't do so the Runtime compatibility will not be insure.
- Automation interfaces
 - ◆ Dassault Systèmes doesn't support Automation interfaces including in CAA V5 code

```
/**  
 * @CAA2Level L1  
 * @CAA2Usage U1  
 */
```

Exercise Presentation → Development Environment

[Student Notes:](#)

And now practice one exercise, to learn about:

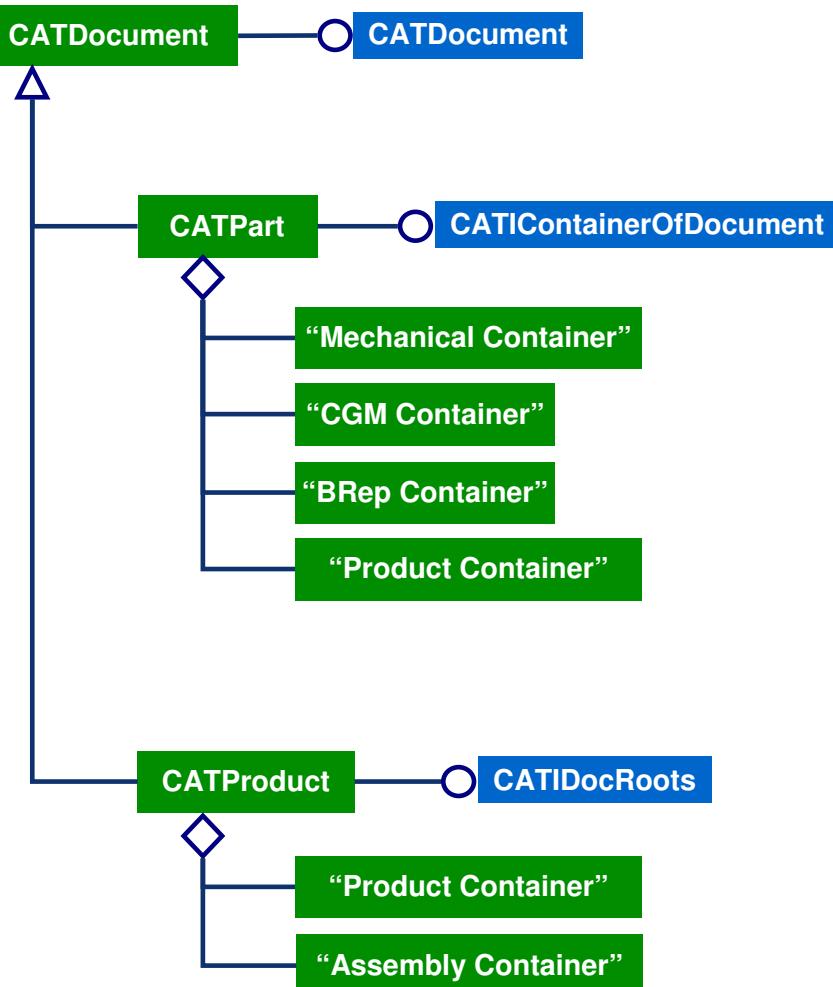
- ◆ **Build a workspace in a shell window**
 - Initialize the environment
 - Build the workspace
 - Launch CATIA

[Student Notes:](#)

Annexes : Interface Summaries

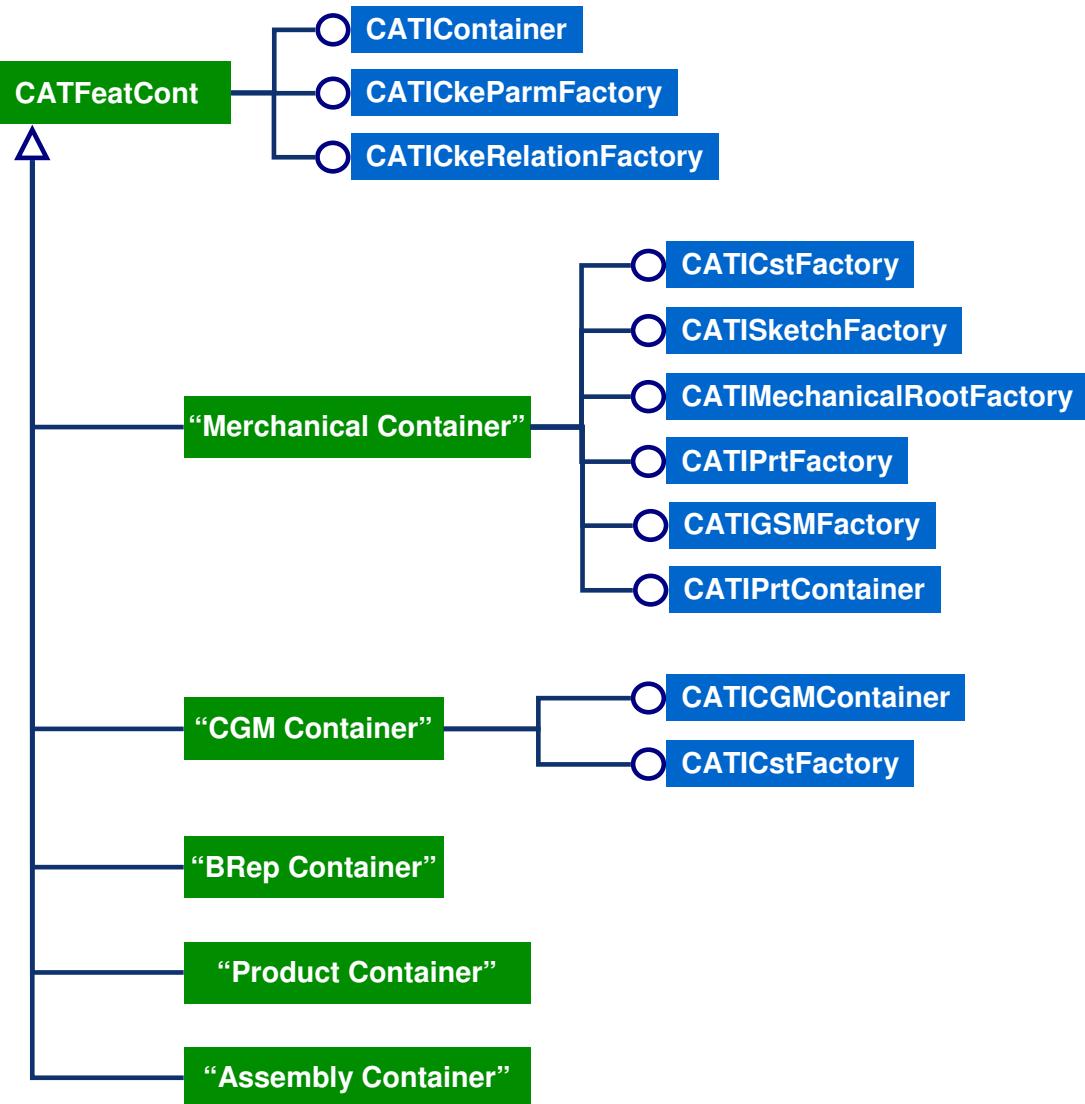
The product and part document

Student Notes:



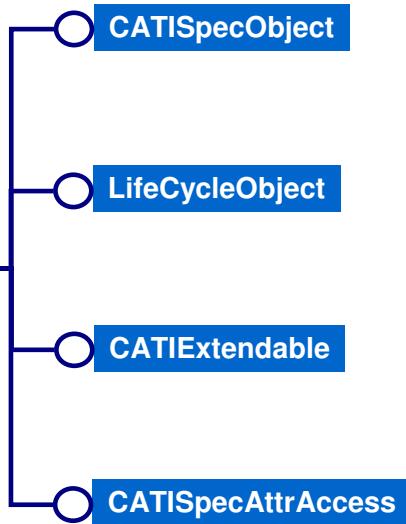
[Student Notes:](#)

Containers



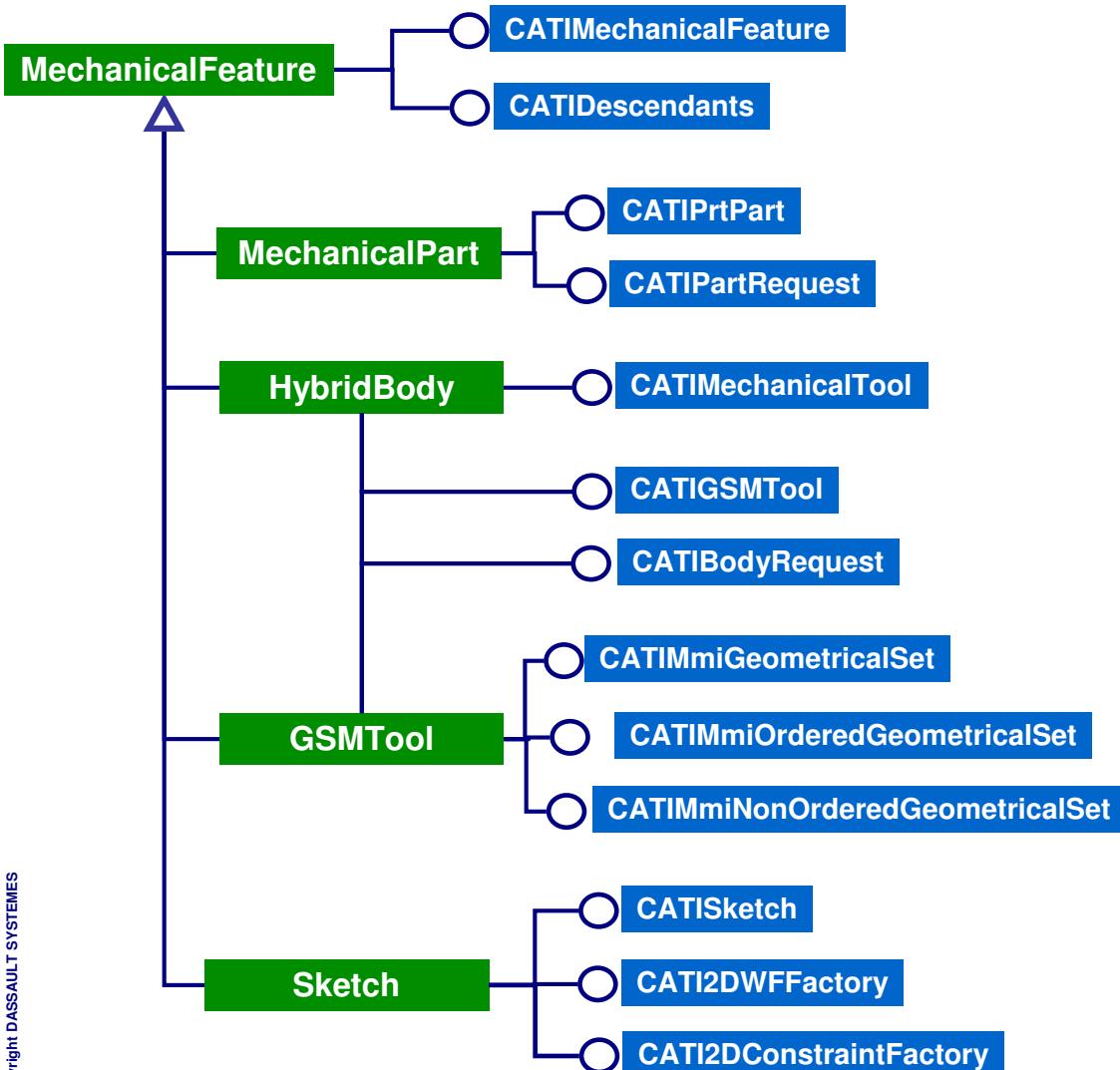
Any features

Student Notes:



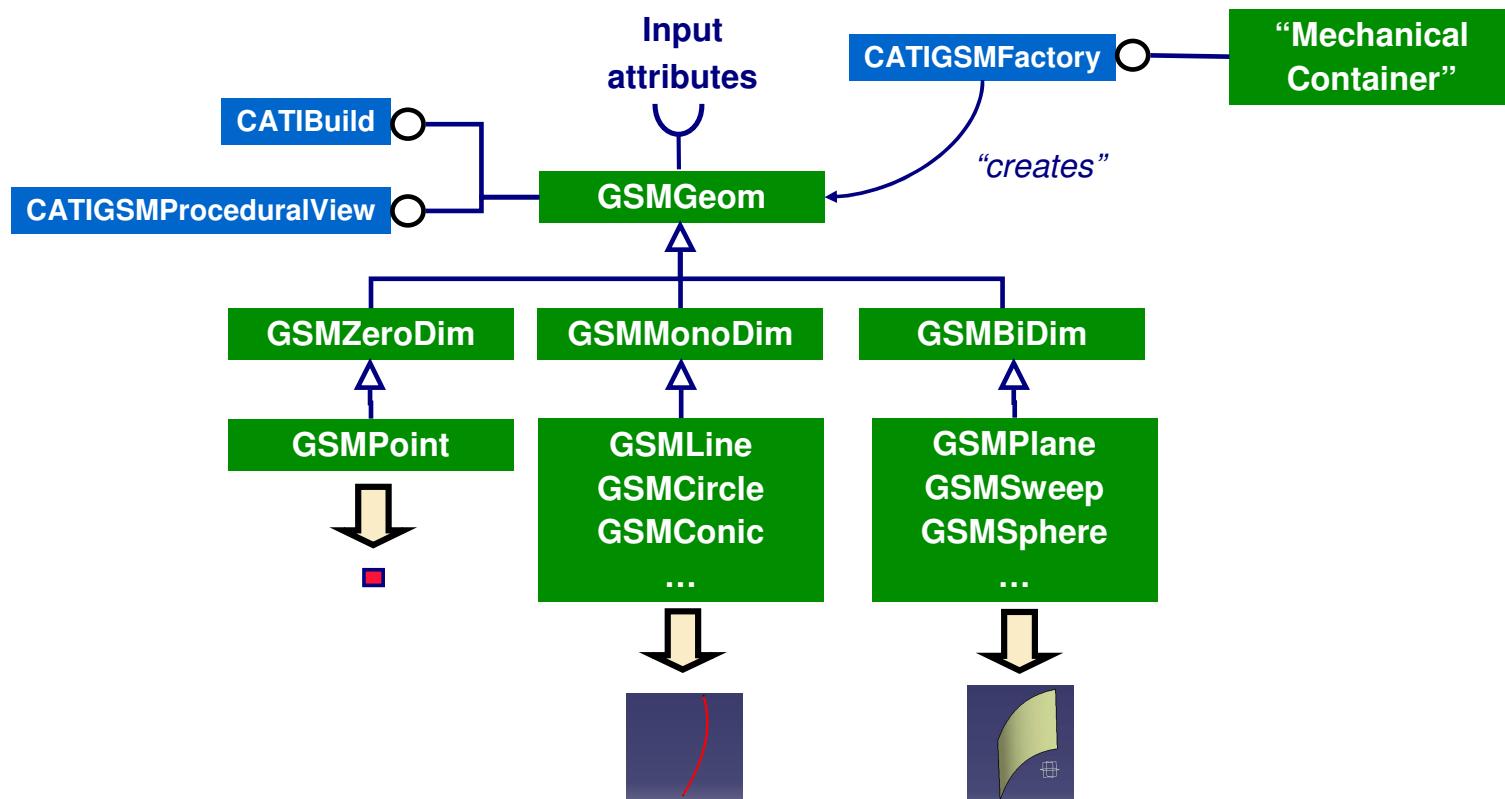
[Student Notes:](#)

Mechanical Root features



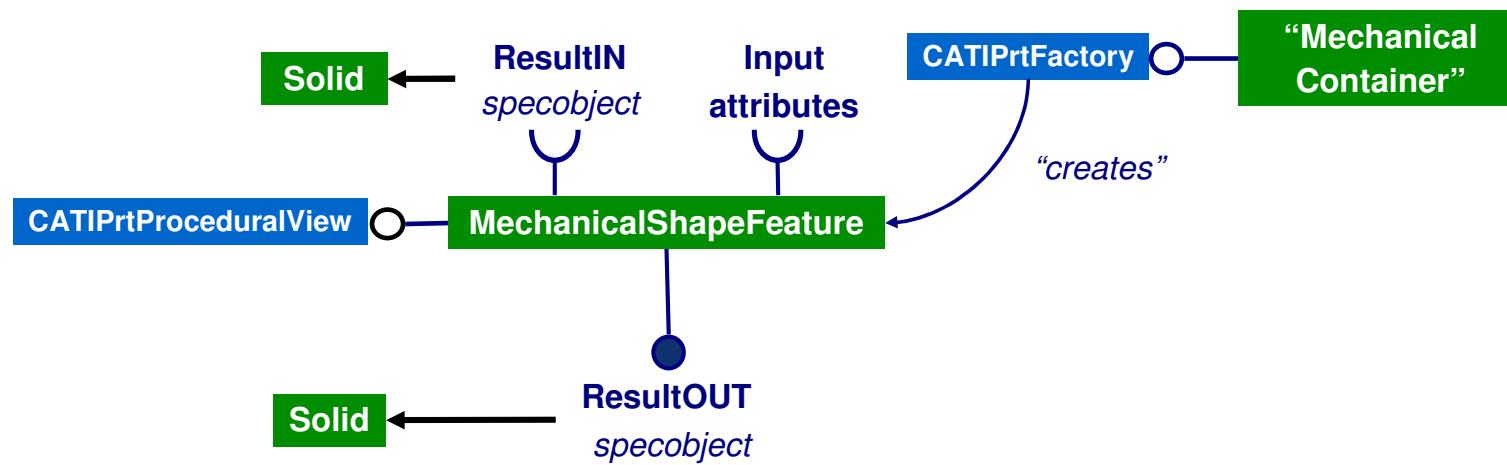
Student Notes:

Surfacic and wireframe features



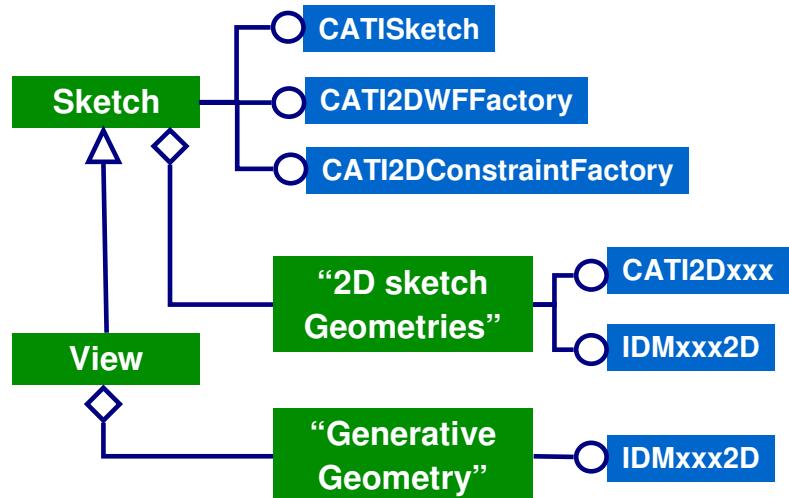
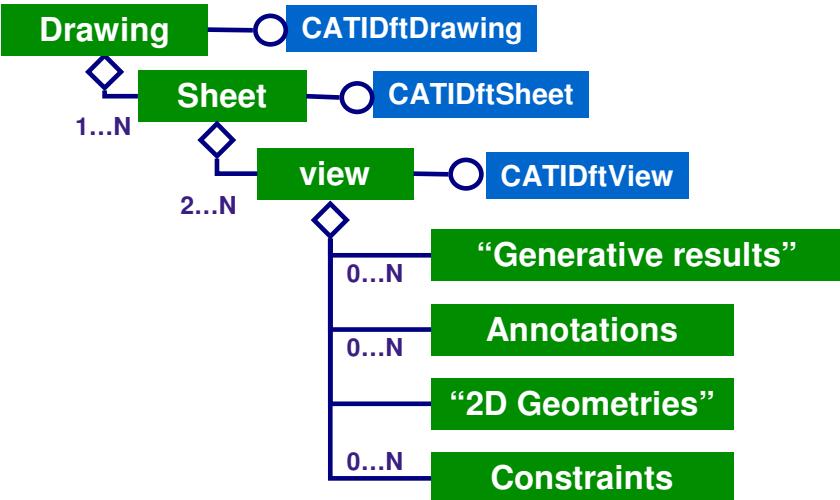
Solid features

Student Notes:



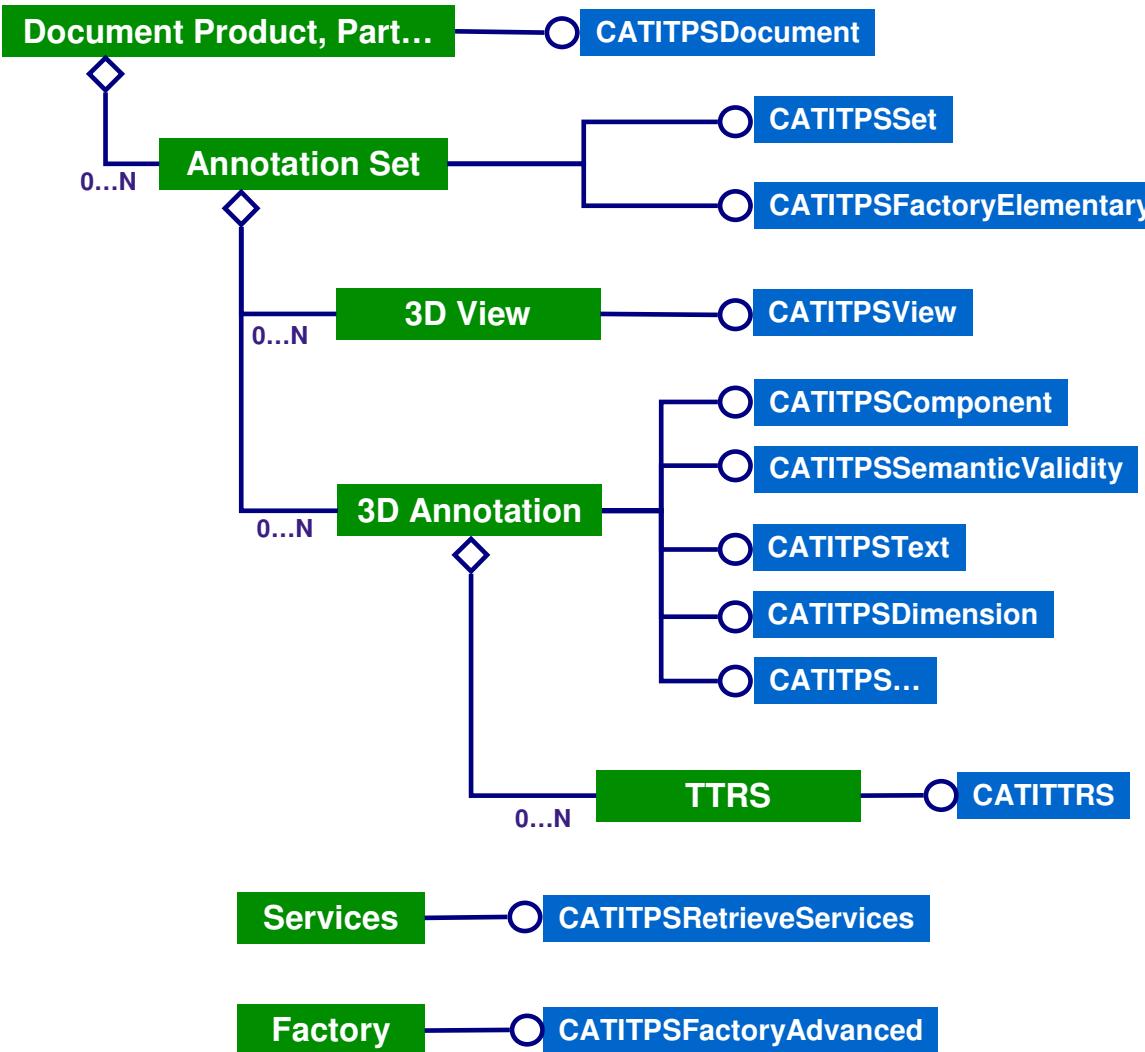
Student Notes:

Drafting features



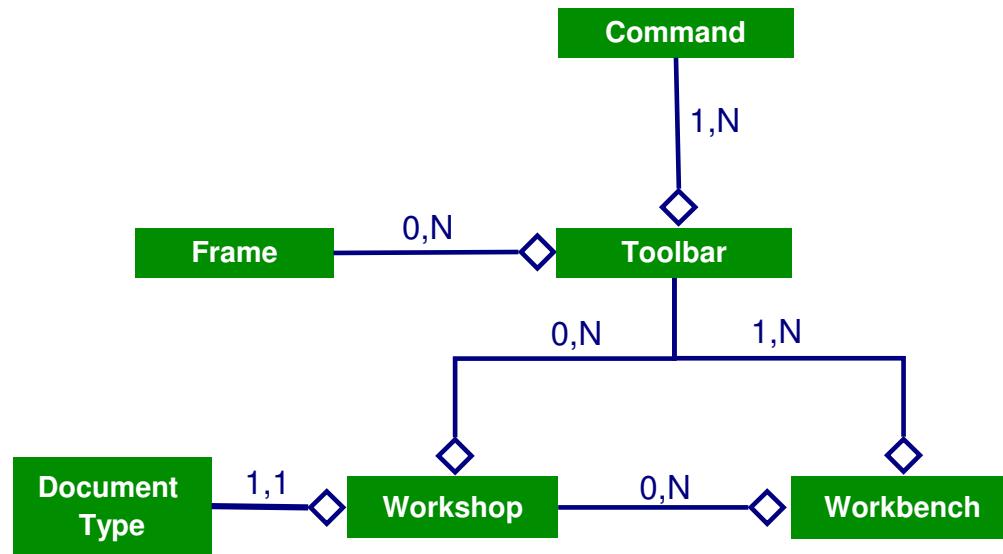
Student Notes:

Tolerancing features



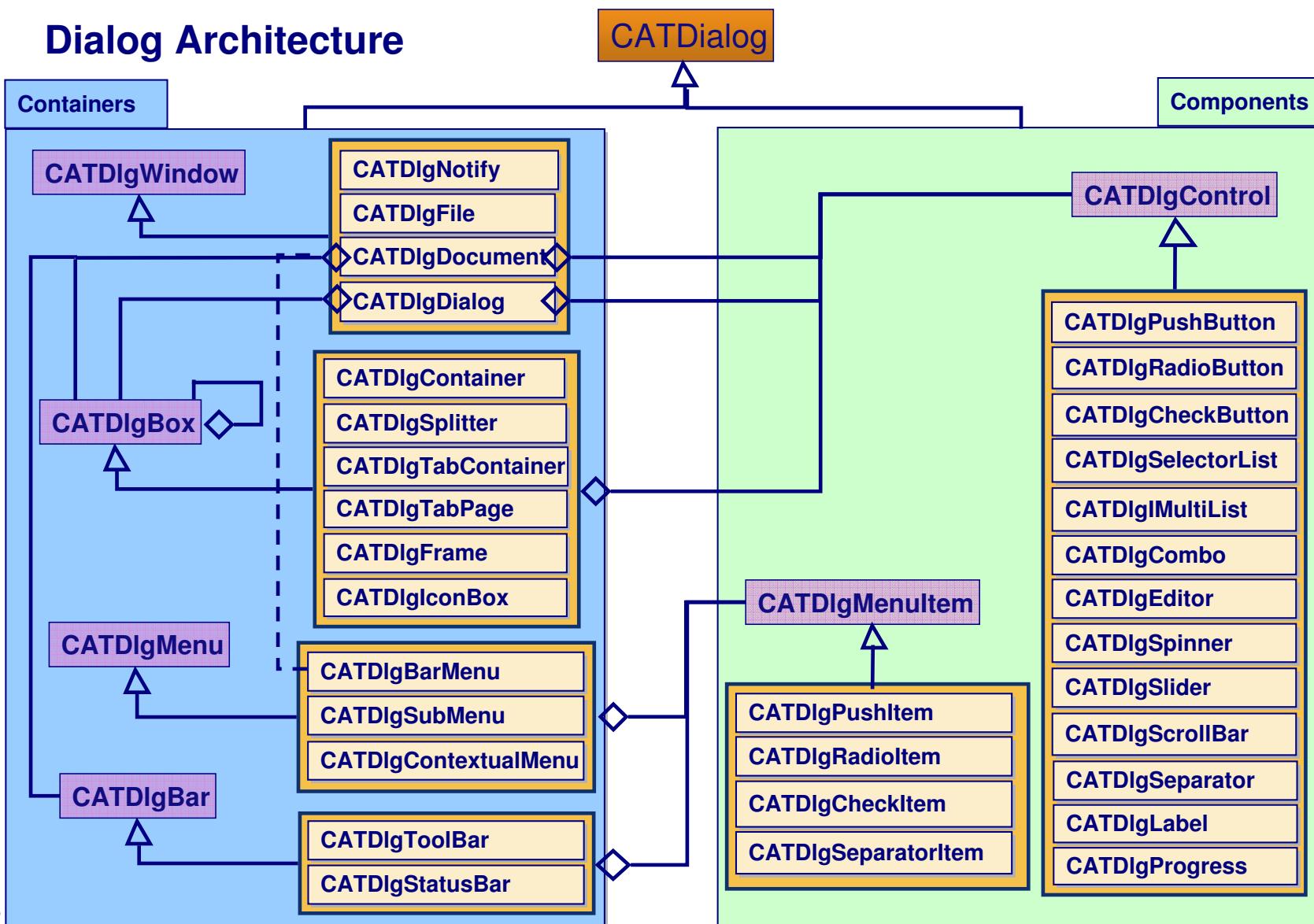
ApplicationFrame architecture

[Student Notes:](#)



[Student Notes:](#)

Dialog Architecture



[Student Notes:](#)

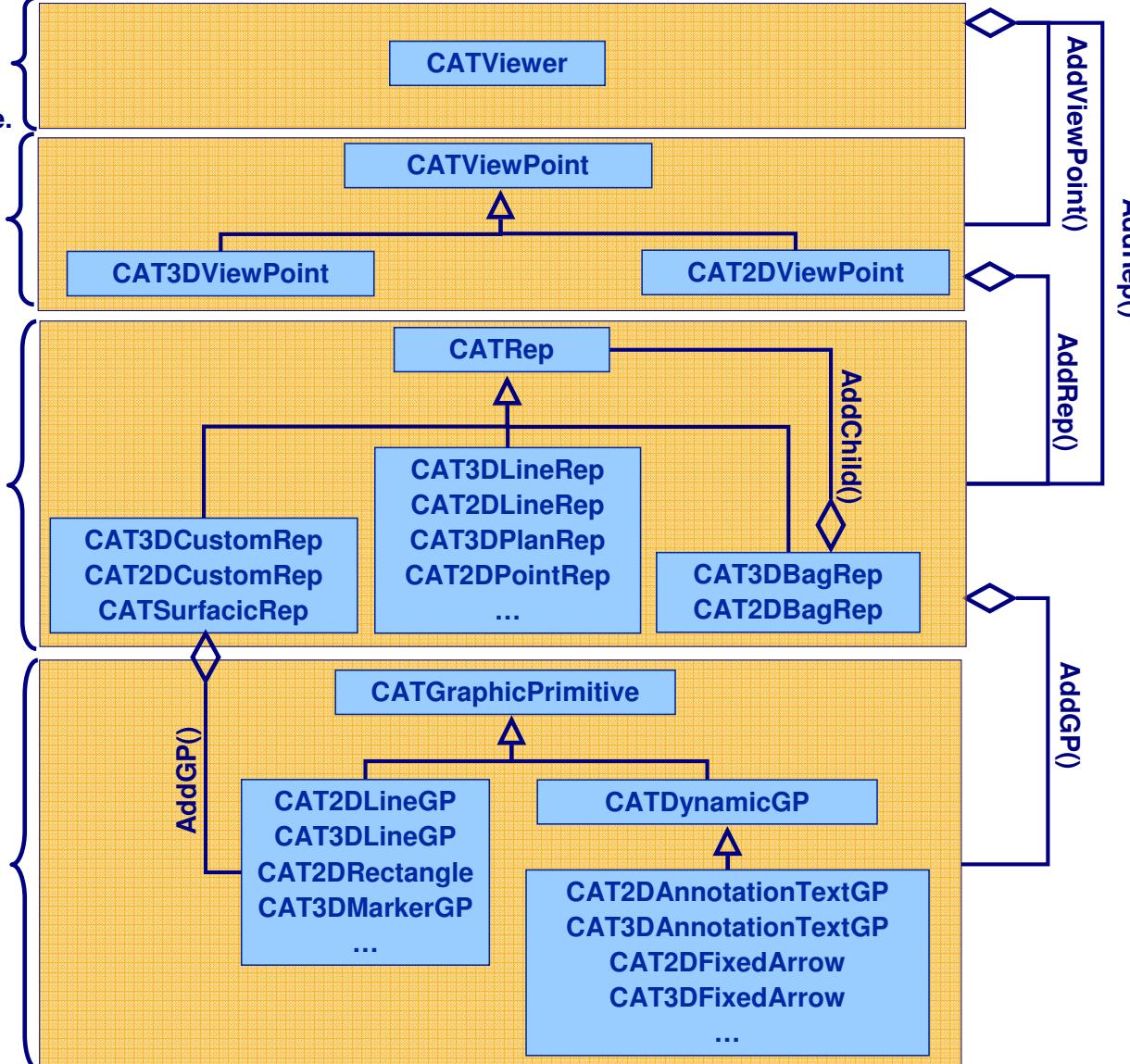
Visualization architecture

VIEWER:
Object allowing to navigate inside the visualized model and to define the visualization mode.

VIEW POINT:
View point of the user on the model

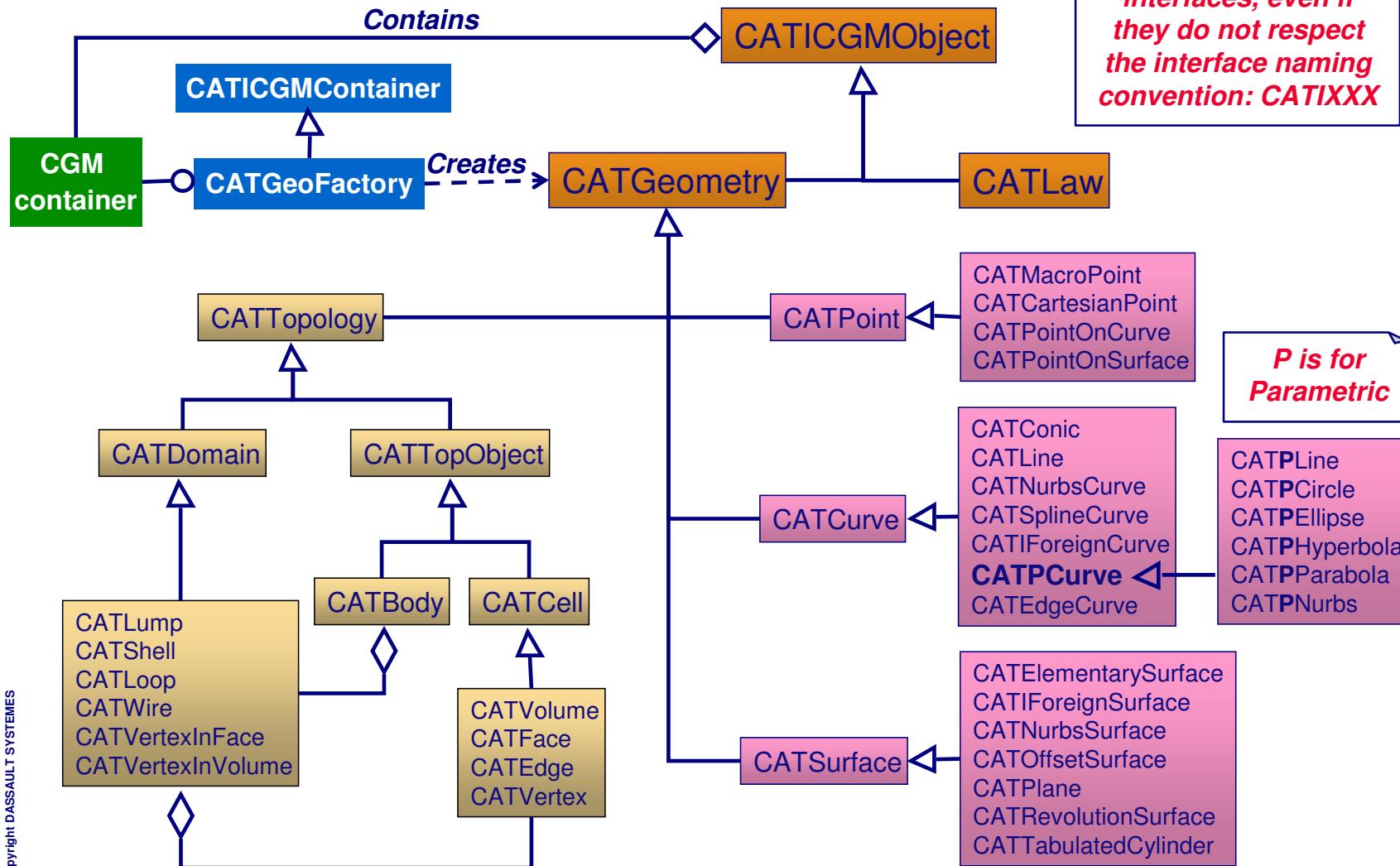
REPRESENTATION:
Define the positioning and the graphic attributes

GRAPHIC PRIMITIVE:
Define the geometry Visualized.



CGM architecture

Student Notes:



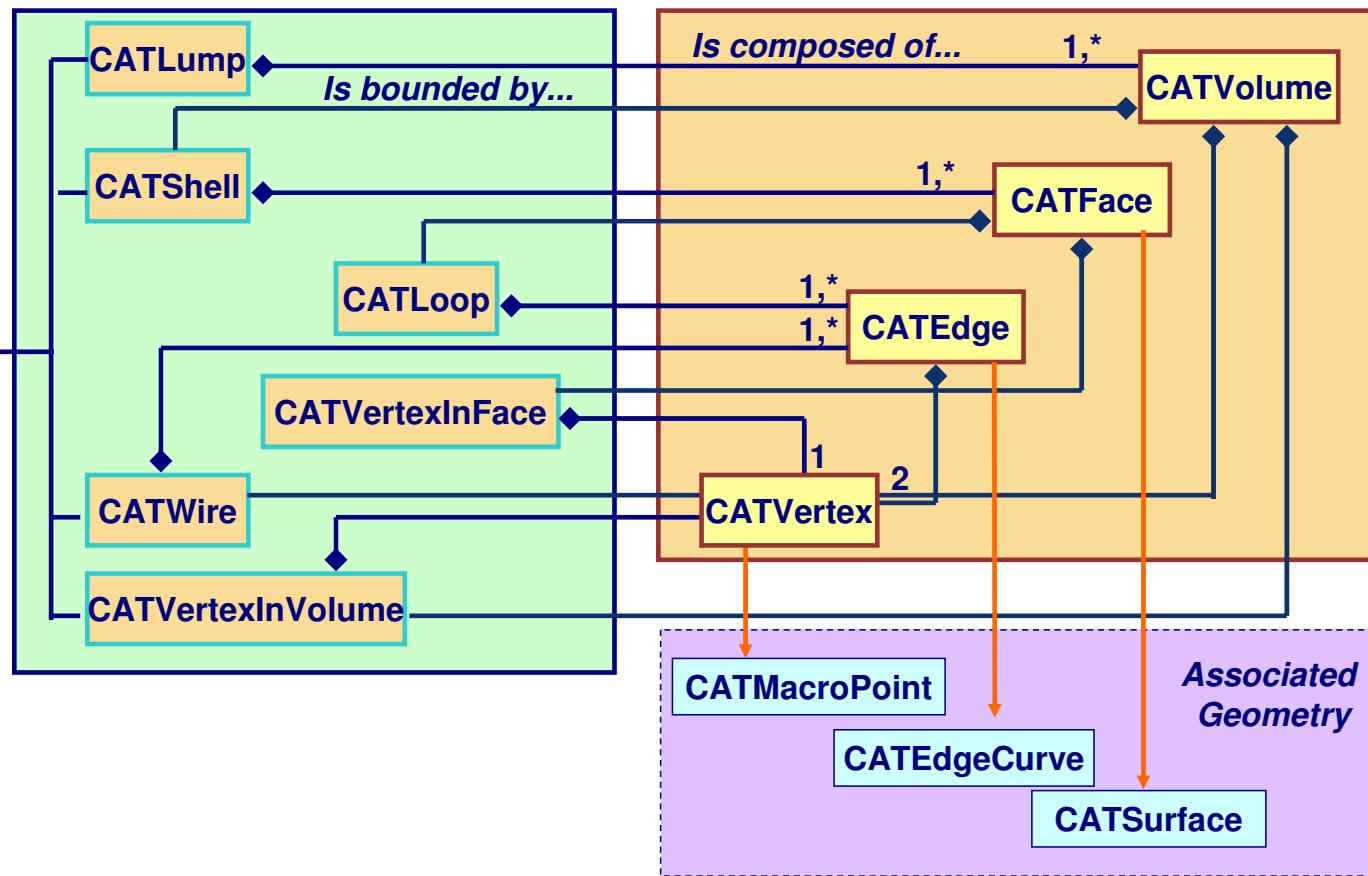
[Student Notes:](#)

Body, Cell and Domain Relationship

CATBody

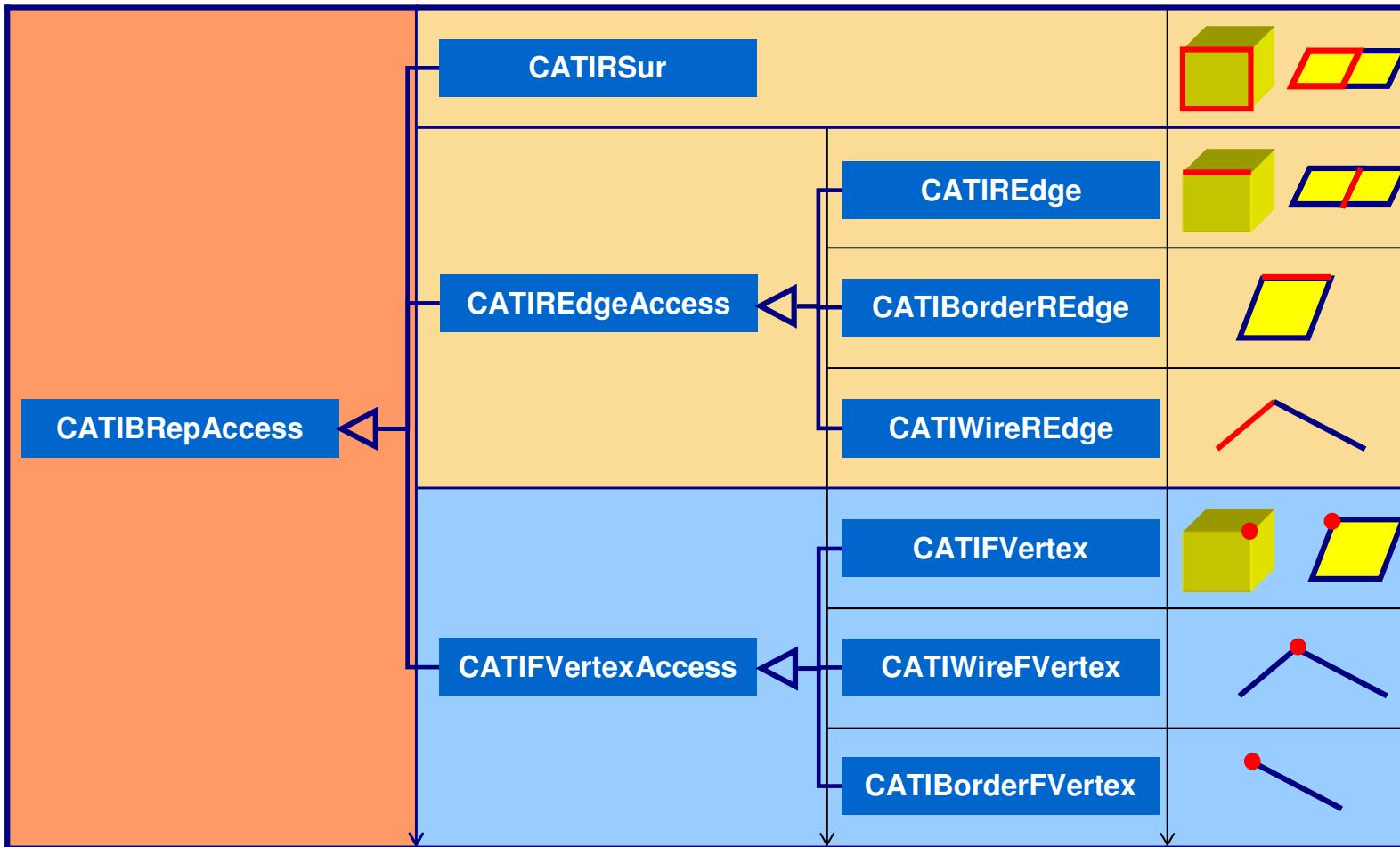
CATDomain

CATCell



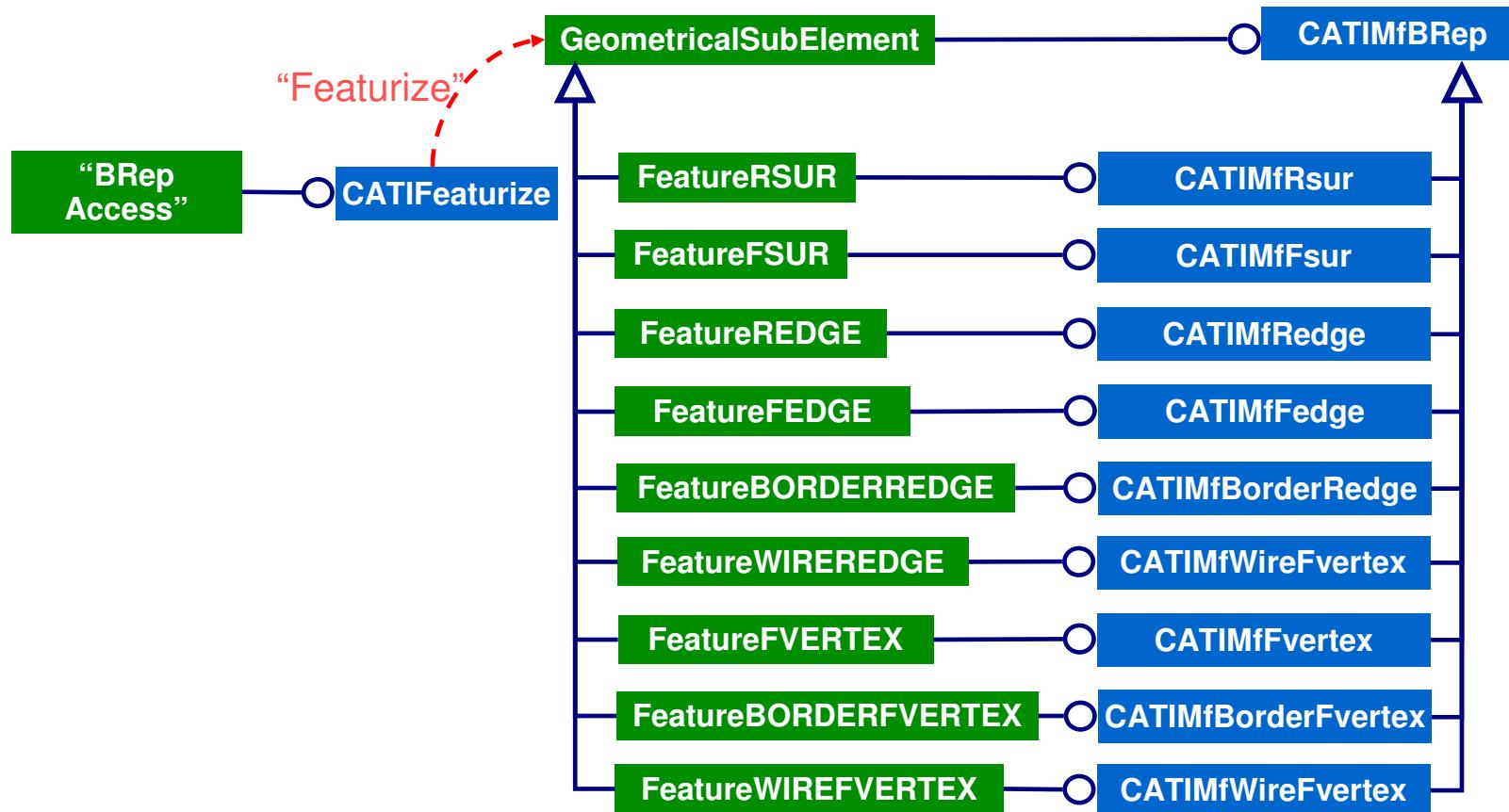
[Student Notes:](#)

BRep Access Interfaces



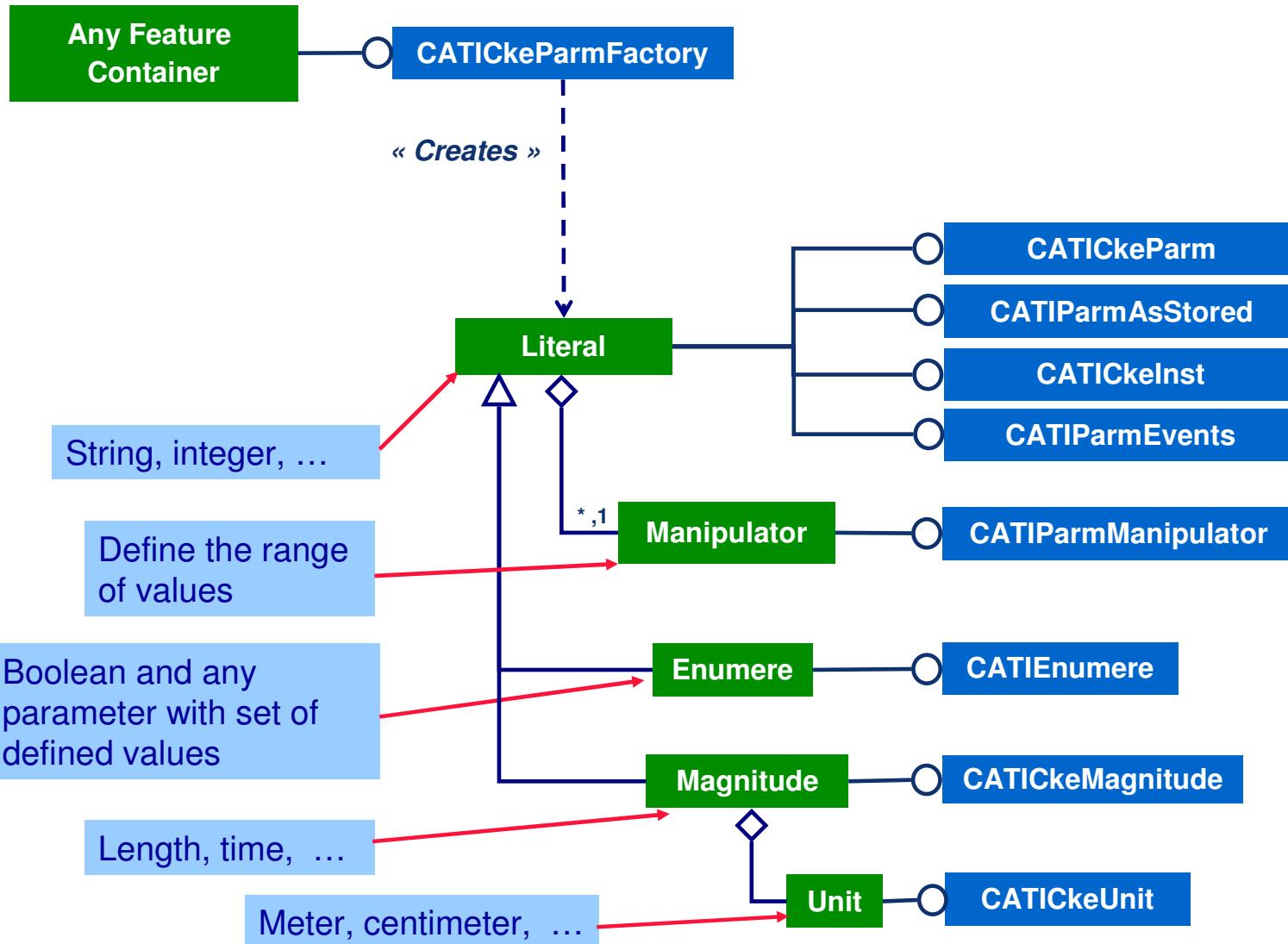
Student Notes:

BRep Feature Architecture



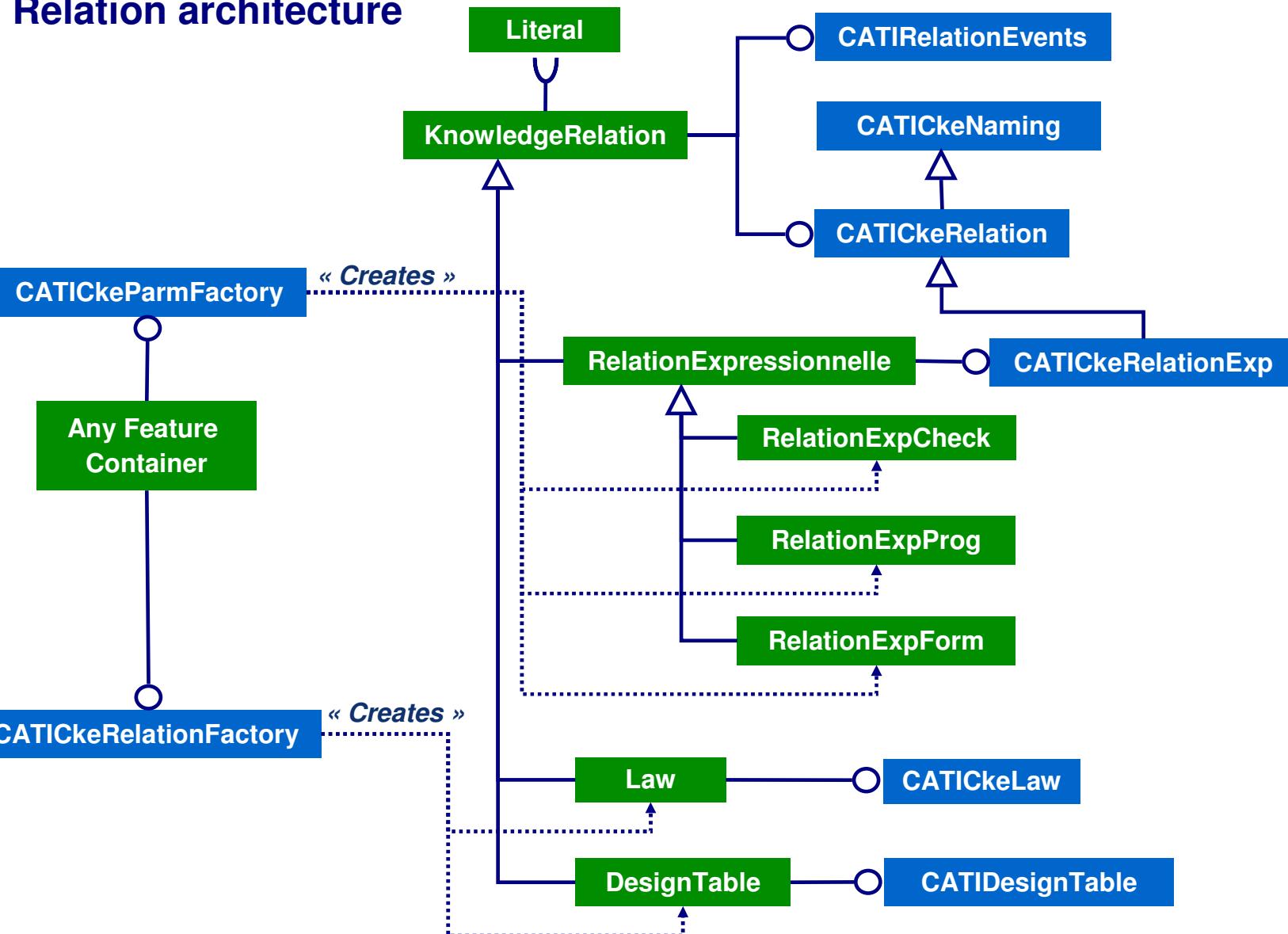
Student Notes:

Literal Features Architecture



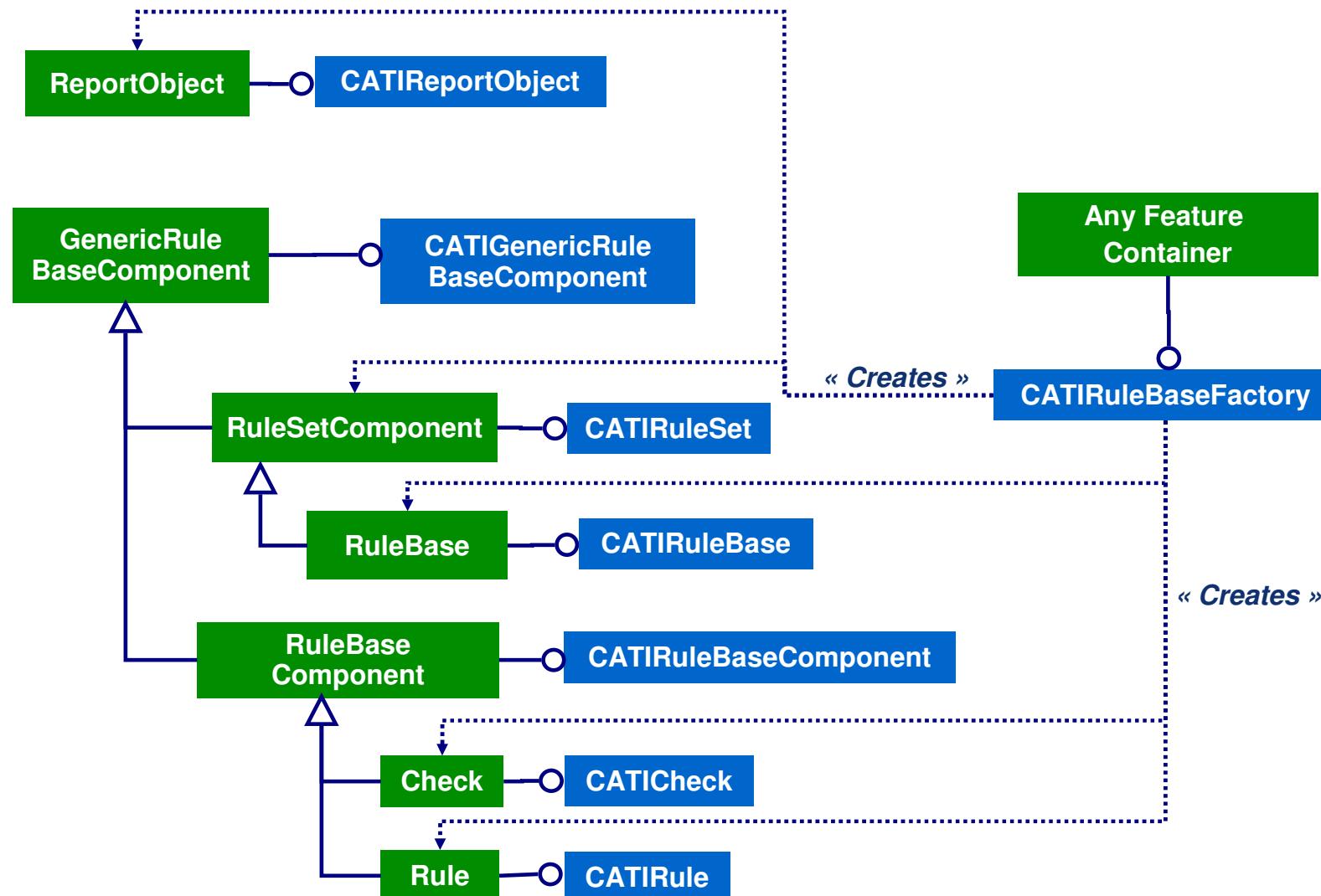
[Student Notes:](#)

Relation architecture



Student Notes:

Knowledge Expert (*KnowHow*) Architecture



Student Notes:

Connectors and publication architecture

