

Descubra o Poder do ObjectARX - Maio 2007

Fernando P. Malard



Fernando Malard é Engenheiro Civil com Mestrado em Engenharia de Estruturas, desenvolvedor ADN, membro do board da AUGI Brasil e Gerente de Desenvolvimento da OFCDesk.

Contato: fernando.malard@augi.com

O AutoCAD se tornou a ferramenta da plataforma CAD mais utilizada nos últimos anos. A maioria dos usuários o utiliza somente para o trabalho do dia a dia e não quer, ou muitas vezes nem tenta, explorar todo seu poder.

Usuários iniciantes e experientes fazem suas pequenas retinas utilizando as programações **AutoLISP** ou **VBA**. Essas plataformas de programação são direcionadas exatamente para esse fim, mas se os usuários quiserem expandir suas capacidades e criar aplicativos realmente poderosos eles precisam usar o **ObjectARX**.

O **ObjectARX** foi lançado oficialmente com o **AutoCAD R13** e se baseia na linguagem **C++**. Exatamente por isso ele assusta os usuários e os coloca afastados dele. O **ObjectARX** é tão poderoso que a própria **Autodesk** o utiliza para criar os chamados Verticais do AutoCAD como o MAP e o ADT. O download do kit de desenvolvimento do **ObjectARX** pode ser feito no site da Autodesk e ele vem com alguns exemplos interessantes. Nesse artigo mostrarei somente uma, mas uma das melhores coisas que podem ser feitas com o **ObjectARX**.

Conceitos Básicos

O núcleo do **AutoCAD** é organizado como um banco de dados aonde você tem locais específicos para guardar cada coisa. Basicamente existem: **Entidades** (objetos com desenho), **Containers** (objetos que armazenam outros objetos) e **Objetos** (objetos sem desenho). O banco de dados tem vários tipos de containers para armazenar layers, blocos, tipos de linhas, etc. A estrutura básica desse banco de dados é mostrada na Figura 1.

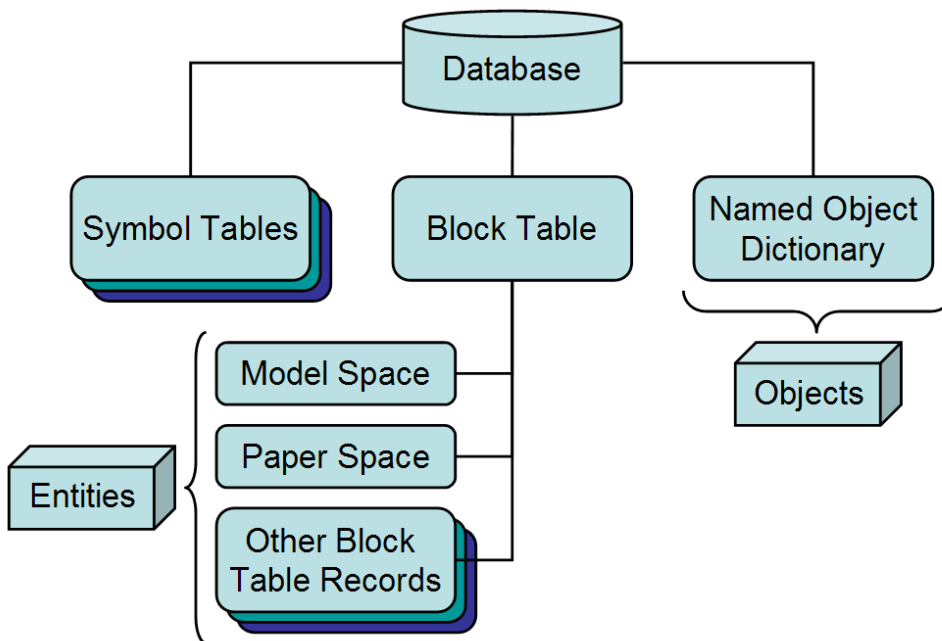


Figura 1 – Estrutura do AutoCAD.

Cada objeto tem seu conjunto de dados e o container apropriado para armazená-lo. Basicamente você deve instanciar o objeto desejado, através de um ponteiro C++, configurar suas propriedades, armazená-lo no container apropriado e fechar seu ponteiro. Essa receita é praticamente a mesma para todos os tipos de objetos. Esse processo, em linhas gerais, será (criando uma entidade LINE):

```
// criar dois pontos e uma linha
AcGePoint3d startPt (1.0, 1.0, 0.0);
AcGePoint3d endPt (10.0, 10.0, 0.0);
AcDbLine* pL =
new AcDbLine (startPt, endPt);
// abrir o container apropriado
AcDbBlockTable* pBT = NULL;
AcDbDatabase* pDB =
acdbHostApplicationServices()->
workingDatabase();
pDB->getSymbolTable(pBT,AcDb::kForRead);
AcDbBlockTableRecord* pBTR = NULL;
pBT->getAt(ACDB_MODEL_SPACE, pBTR, AcDb::kForWrite);
pBT->close();
// agora, adicionar a entidade ao container
AcDbObjectId Id;
pBTR->appendAcDbEntity(Id, pL);
pBTR->close();
pL->close();
```

Esse código criará uma entidade **LINE** do ponto **(1,1,0)** ao ponto **(10,10,0)** no layer atual.

Visão geral do aplicativo

Os aplicativos **ObjectARX** são basicamente compostos de uma ou mais módulos **DLL**. Os módulos **DLL** podem usar **MFC**, **Win32** puro ou até **.NET**. De fato, o ObjectARX separa o conceito de interface e classes para permitir a criação dos chamados "object enablers" para seus aplicativos. O módulo de Interface tem a extensão de arquivo **ARX** e o módulo de classe a extensão **DBX**. Obviamente, aplicativos complexos irão requerer uma estrutura de projeto muito mais sofisticada.

Seu aplicativo poderá registrar comandos, criar novas classes e proporcionar uma completa integração com a interface do **AutoCAD**. Uma vez pronto, seu aplicativo pode ser carregado no **AutoCAD** usando o comando **APpload** ou **ARX**. Uma vez carregado, ele pode se comunicar com o banco de dados do AutoCAD's ou sua interface, usar o prompt de comandos ou exibir algumas janelas **MFC**.

O processo de criação de um projeto **ObjectARX** não é muito simples e para facilitar o kit de desenvolvimento contém o **ARXWizard**. Esse assistente irá ajudá-lo a construir o chamado "esqueleto" do projeto poupando seu tempo. Para criar aplicativos **ObjectARX** compatíveis com o **AutoCAD 2007** ou **2008** você irá usar o **Visual Studio .NET 2005**.

Criando sua Entidade

Agora vamos ver como criar sua própria entidade o que lhe dará uma boa visão do potencial do **ObjectARX**. Vamos usar o **ARXWizard** para simplificar o processo e omitiremos alguns comentários e procedimentos de segurança no código para mantê-lo simples e sucinto.

Usando o **VS2005**, crie uma **Blank Solution** chamada **CustomEntitySample**. Depois adicione **dois** projetos ObjectARX. O primeiro será um módulo **ARX** chamado **CustEntityARX** e o segundo será um módulo **DBX** chamado **CustEntityDBX**. Marque **MFC extensions** em ambos. Faça o projeto **ARX dependente do DBX** clicando com o botão direito no projeto ARX, selecione "**Project Dependencies**" e marque o projeto DBX na lista. Agora na toolbar do **ARXWizard** clique no botão **Autodesk Class Explorer**. Clique com o botão direito no nó **CustEntityDBX** e selecione "**Add an ObjectDBX Custom Object...**" (Figura 2)

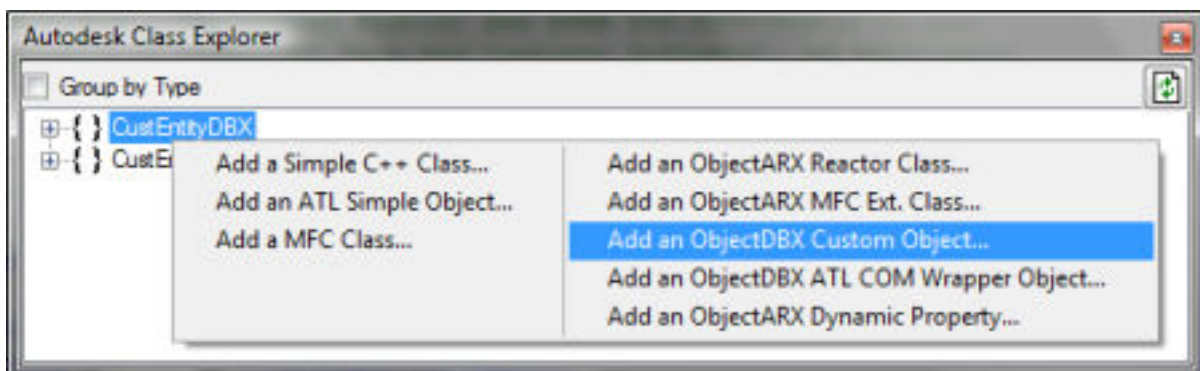


Figura 2 – Autodesk Class Explorer.

Uma janela aparecerá pedindo informações da sua classe (Names page). Digite **MyCustomEntity** como nome da classe e escolha **AcDbEntity** como classe base. Vá à página Protocols e marque **DWG**, **Osnap** e **Grip-point** protocols. Finalize clicando em Finish. Agora você poderá compilar seu projeto sem erros.

Abra novamente o **Autodesk Class Explorer** e vá ao projeto **CustEntityDBX**. Selecione o nó **MyCustomEntity**, clique com o botão direito nele e selecione **"Add Variable..."**. A janela **Member Variable Wizard** aparecerá (Figura 3)

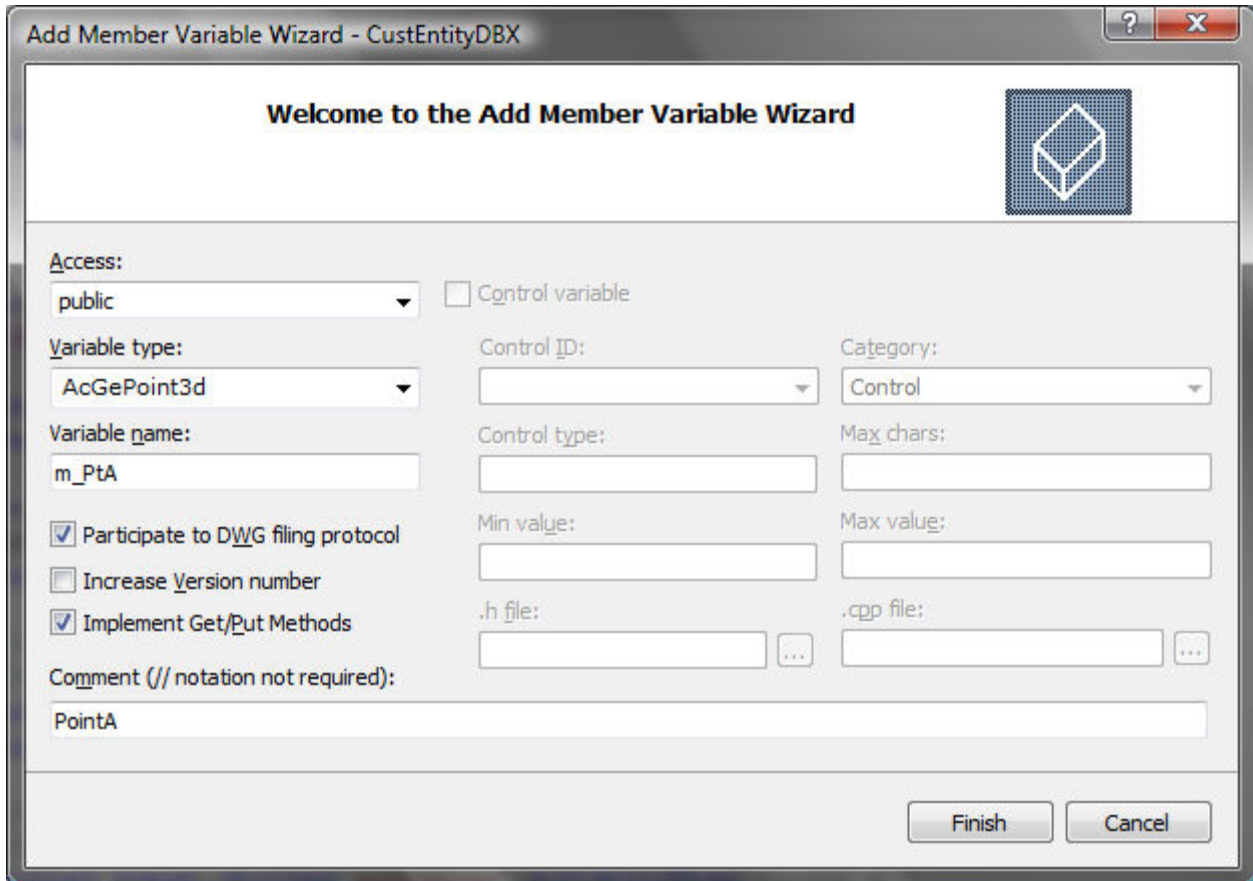


Figura 3 – Member Variable Wizard.

Vamos adicionar quatro variáveis do tipo **AcGePoint3d** para construir a representação gráfica da nossa entidade que será um retângulo. Os nomes das variáveis podem ser: **m_PtA**, **m_PtB**, **m_PtAB** e **m_PtBA**. Marque a opção **"Participate to DWG filing protocol"**, desmarque **"Increase Version number"**, marque **"Implement Get/Put methods"** e preencha os comentários para cada uma.

Desenho da Entidade

O desenho da entidade é feito através dos métodos **worldDraw()** e/ou **viewportDraw()**. Nesse caso, somente usaremos o método **worldDraw()** pois não teremos gráficos dependentes de viewports. Localize esse método no arquivo **MyCustomEntity.CPP** e adicione o código:

```

Adesk::Boolean MyCustomEntity::worldDraw (AcGiWorldDraw *mode) {
assertReadEnabled();
// Polyline de contorno, marca 1, cor vermelha
AcGePoint3d pts[4];
pts[0] = m_PtA;

```

```

pts[1] = m_PtAB;
pts[2] = m_PtB;
pts[3] = m_PtBA;

mode->
subEntityTraits().setSelectionMarker(1);
mode->subEntityTraits().setColor(1);
mode->geometry().polygon(4,pts);

// Texto, marca 2, cor bylayer
mode-> subEntityTraits().setSelectionMarker(2);
mode->subEntityTraits().setColor(256);

AcGiTextStyle style;
style.setFileName(_T("txt.shx"));
style.setBigFontFileName(_T(""));
style.setTextSize(25);
style.loadStyleRec();

AcGePoint3d txtPt ((m_PtB.x+m_PtA.x) / 2.0, (m_PtB.y+m_PtA.y) / 2.0, m_PtA.z);

CString strText = _T("Any text");
mode->
geometry().text(txtPt, AcGeVector3d::kZAxis,
(m_PtAB-m_PtA), strText, strText.GetLength(), Adesk::kFalse, style);

return Adesk::kTrue;
}

```

Pontos de GRIP

Nossa entidade possuirá ponto de ação GRIP. Os grips são implementados pelo método **getGripPoints()** e suas ações especificadas pelo método **moveGripPointsAt()**. Localize esses métodos e adicione o código seguinte:

```

Acad::ErrorStatus
MyCustomEntity::getGripPoints (
AcGePoint3dArray &gripPoints,
AcDbIntArray &osnapModes,
AcDbIntArray &geomIds) const
{
assertReadEnabled ();
gripPoints.append(m_PtA);
gripPoints.append(m_PtAB);
gripPoints.append(m_PtB);
gripPoints.append(m_PtBA);
gripPoints.append(AcGePoint3d((m_PtB.x+m_PtA.x)/2.0,(m_PtB.y+m_PtA.y)/2.0,m_PtA.z))
;
return Acad::eOk;
}

Acad::ErrorStatus MyCustomEntity::moveGripPointsAt (
const AcDbIntArray &indices,
const AcGeVector3d &offset)
{
assertWriteEnabled ();

```

```

for(int i=0;i<indices.length();i++) {
    int idx = indices.at(i);
    if (idx==0 || idx==4) m_PtA += offset;
    if (idx==1 || idx==4) m_PtAB += offset;
    if (idx==2 || idx==4) m_PtB += offset;
    if (idx==3 || idx==4) m_PtBA += offset;
}
return Acad::eOk;
}

```

Para habilitar os GRIP points você só precisa adicionar os pontos no array recebido pela função. Para aplicar uma ação ao GRIP **aplique o vetor de transformação** para cada ponto selecionado. Como as entidades podem conter diversos pontos eles são indexados pela ordem de sua adição ao array e o mesmo índice deve ser considerado ao aplicar a respectiva ação.

Pontos de OSNAP

Outra funcionalidade importante que você pode desejar está relacionada com a precisão. O Object Snap permite aos usuários selecionar pontos precisos na sua entidade. Nesse exemplo vamos implementar 3 pontos de Object Snap: **EndPoint**, **MidPoint** e **Center**. Isso é feito através do método **getOsnapPoints()**:

```

Acad::ErrorStatus
MyCustomEntity::getOsnapPoints (
AcDb::OsnapMode osnapMode,
int gsSelectionMark,
const AcGePoint3d &pickPoint,
const AcGePoint3d &lastPoint,
const AcGeMatrix3d &viewXform,
AcGePoint3dArray &snapPoints,
AcDbIntArray &geomIds) const
{
    assertReadEnabled();
    switch (osnapMode) {
    case AcDb::kOsModeEnd:
        snapPoints.append(m_PtA);
        snapPoints.append(m_PtAB);
        snapPoints.append(m_PtB);
        snapPoints.append(m_PtBA);
        break;
    case AcDb::kOsModeMid:
        snapPoints.append(m_PtA + ((m_PtAB - m_PtA ).length() / 2.0) * ((m_PtAB - m_PtA
        ).normalize()));
        snapPoints.append(m_PtAB + ((m_PtB - m_PtAB).length()/2.0)*((m_PtB - m_PtAB
        ).normalize()));
        snapPoints.append(m_PtB + ((m_PtBA - m_PtB ).length()/2.0)*((m_PtBA - m_PtB
        ).normalize()));
        snapPoints.append(m_PtBA + ((m_PtA - m_PtBA).length()/2.0)*((m_PtA - m_PtBA
        ).normalize()));
        break;
    case AcDb::kOsModeCen:
        snapPoints.append(AcGePoint3d(
        (m_PtB.x+m_PtAB.x+m_PtBA.x+m_PtA.x)/4.0,(m_PtB.y+m_PtAB.y+m_PtBA.y+m_PtA.y)/
        4.0, m_PtA.z));
    }
}

```

```
break;
}
return Acad::eOk;
}
```

Para programar o **EndPoint** apenas adicionamos os 4 pontos da entidade. Já para o **MidPoint** e **Center** precisamos fazer alguns cálculo geométricos.

Transformações

Transformações podem ser aplicadas na nossa entidade e isso é feito através do método **transformBy()**. Abra o **Autodesk Class Explorer** novamente, selecione o nó **MyCustomEntity** e expanda a classe base (**AcDbEntity**). Procure pelo método **transformBy()** e acione o botão direito sobre ele. Selecione "Implement Base Class Method". O assistente irá adicionar esse método na classe. Abra o arquivo **CPP** e acrescente o código:

```
Acad::ErrorStatus
MyCustomEntity::transformBy(
const AcGeMatrix3d & xform)
{
assertWriteEnabled();
m_PtA.transformBy(xform);
m_PtAB.transformBy(xform);
m_PtB.transformBy(xform);
m_PtBA.transformBy(xform);
return Acad::eOk;
}
```

Nesse caso só precisamos aplicar a mesma matriz de transformação recebida para os 4 pontos. O resultado vai ser a transformação completa da entidade.

Projeto ARX

Finalmente, vamos criar a nossa entidade de dentro do módulo **ARX**. A classe da entidade é exportada do módulo **DBX** e é isso que vai permitir a sua utilização no módulo **ARX**. Para que o projeto **ARX** reconheça essa classe precisamos incluir o arquivo de declaração da classe no arquivo **StxAfx.h** do projeto **ARX**. Abra esse arquivo e adicione a inclusão seguinte no fim do arquivo:

```
#include "..\CustEntityDBX\MyCustomEntity.h"
```

Agora, na árvore do **Solution Explorer**, selecione o projeto **ARX** e clique no ícone "a>" na toolbar do ARXWizard. Botão direito na lista superior, selecione New. Use o nome **MYCUSTENT** com o modo **Modal**. O **ARXWizard** irá adicionar, dentro do arquivo **acrEntryPoint.cpp**, um novo método chamado **CustEntityARX_MyCustEnt** dentro da classe **CCustEntityARXApp**. Acrescente o código seguinte:

```
static void CustEntityARX_MyCustEnt(void) {
// Input information
ads_point pt1,pt2;
if (acedGetPoint(NULL,_T("Primeiro canto:\n"), pt1) != RTNORM) return;
if (acedGetCorner(pt1,_T("Segundo canto:\n"), pt2) != RTNORM) return;
MyCustomEntity *pEnt =
new MyCustomEntity();
```

```

pEnt->put_m_PtA (asPnt3d(pt1));
pEnt->put_m_PtAB (AcGePoint3d (pt2[X], pt1[Y], pt1[Z]));
pEnt->put_m_PtB (asPnt3d(pt2));
pEnt->put_m_PtBA (AcGePoint3d (pt1[X], pt2[Y], pt2[Z]));
// Post to Database
AcDbBlockTable *pBlockTable;
acdbHostApplicationServices()->
workingDatabase()->getSymbolTable (pBlockTable, AcDb::kForRead);
AcDbBlockTableRecord *pBlockTableRecord;
pBlockTable->getAt(ACDB_MODEL_SPACE, pBlockTableRecord,AcDb::kForWrite);
pBlockTable->close();
AcDbObjectId retId = AcDbObjectId::kNull;
pBlockTableRecord->appendAcDbEntity(retId, pEnt);
pBlockTableRecord->close();
pEnt->close();
}

```

Esse processo é quase o mesmo feito para a entidade **AcDbLine** mas dessa vez estamos usando nossa própria classe e nossos próprios métodos.

Executando o aplicativo

Agora você deve ser capaz de compilar o projeto sem nenhum erro. Depois de compilar, abra o **AutoCAD** e acione o comando **APLOAD**. Carrega primeiro o módulo **DBX** e em seguida o módulo **ARX**. Acione o comando **MYCUSTENT** e crie algumas entidades. Teste os comandos **MOVE**, **ROTATE**, acione os **GRIP** e utilize os pontos de precisão com o **OSNAP** (Figura 4).

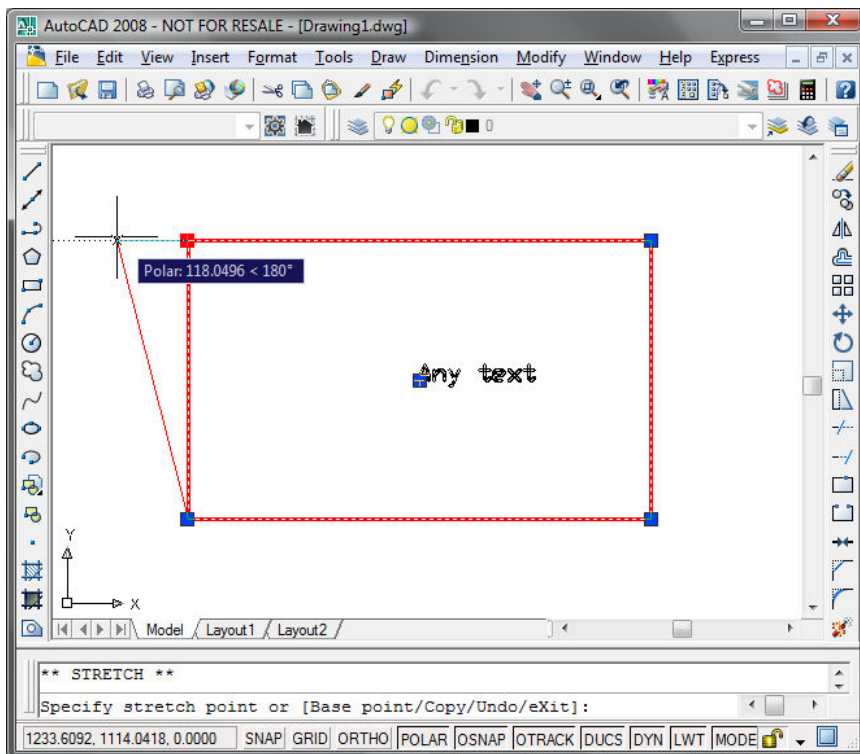


Figura 4 – Entidade no AutoCAD.



Conclusão

Esse artigo foi criado para mostrar somente uma introdução ao poder do **ObjectARX**. Esse é apenas o começo de grandes coisas que você poderá criar usando o **ObjectARX**. Existem várias outras funcionalidades que você pode adicionar aos seus aplicativos e isso permitirá que você crie grandes soluções para diversas áreas da Indústria.