ФГБОУВПО «Воронежский государственный технический университет»

кафедра компьютерных интеллектуальных технологий проектирования

268-2011

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к практическим работам по дисциплине «Разработка САПР» для студентов направления 230100.64 профиля «Системы автоматизированного проектирования в машиностроении» очной формы обучения

Воронеж 2011

Составители: канд. техн. наук А.Н. Юров,
канд. техн. наук М.В. Паринов,
д-р. техн. наук М.И. Чижов,
ст. преп. В.А. Рыжков

УДК 621.87.621.357.74

Методические указания к практическим работам по дисциплине «Разработка САПР» для студентов направления 230100.64 профиля «Системы автоматизированного проектирования в машиностроении» очной формы обучения. / ФГБОУ ВПО «Воронежский государственный технический университет»; сост. А.Н. Юров, М.В. Паринов, М.И. Чижов, В.А. Рыжков. Воронеж, 2011. 34 с.

Методические указания содержат практический материал по разработке прикладных библиотек в визуальных средах проектирования с использованием языка высокого уровня C# с использованием средств автоматизации интерактивной системы моделирования NX, приводятся примеры разработки сборочного узла на каждой стадии программного моделирования.

Предназначены для студентов 3 курса.

Методические указания подготовлены в электронном виде в текстовом редакторе MS Word 2007 и содержатся в файле му_nxopen.doc.

Ил. 9. Библиогр.: 5 назв.

Рецензент д-р техн. наук, проф. В.Н. Старов

Ответственный за выпуск зав. кафедрой д-р техн. наук, проф. М.И. Чижов

Издается по решению редакционно-издательского совета Воронежского государственного технического университета

ВВЕДЕНИЕ

Система автоматизации NX оснащена пакетом программируемого интерфейса для разработчика (NX Open API), позволяющая ускорить разработку и выпуск готовых изделий посредством использования дополнительных модулей на основе гибкой системы иерархии создания и управления объектами. Актуальность использования таких программных решений основывается, прежде всего, на идеологии повторяемости тех или иных операций при проектировании и создании требуемого конструкторского решения, а также сложностью проектных работ и узлов агрегатов, требующих значительных временных затрат на производстве. Открытость программного интерфейса NX Open API позволяет динамично реагировать на все изменения процессов жизненного цикла изделий машиностроительного профиля, а интеграция с системой посредством родного или стороннего интерфейсов обеспечивают создание законченных дополнительных приложений для решения целевых задач современных производственных подразделений. В предлагаемой работе будет проведено исследование по созданию узла заготовительной оснастки средствами встроенного программного интерфейса NX API посредством использования прикладного языка разработки C# (Си Шарп) в среде Microsoft Visual Studio 2010. Предложенные методики, благодаря концепции объектно-ориентированного подхода, можно перенести на иные платформы и средства разработки посредством языка C++ кроссплатформенной IDE Qt Creator с фреймворком QT, разработанной TrollTech (Nokia) или свободной интегрированной среды разработки Eclipse.

ОСНОВНЫЕ ПОЛОЖЕНИЯ ПРИ ПРОЕКТИРОВАНИИ СБОРОЧНЫХ УЗЛОВ СРЕДСТВАМИ ВИЗУАЛЬНЫХ СРЕД РАЗРАБОТКИ В ИНТЕРАКТИВНОЙ СИСТЕМЕ АВТОМАТИЗАЦИИ ИНСТРУМЕНТАЛЬНЫМИ РЕШЕНИЯМИ NX OPEN API

ПОРЯДОК ВЫПОЛНЕНИЯ СБОРОЧНОГО КОМПОНЕНТА

Построение сборочного узла на примере фиксатора (рис.1), входящего в состав узлов заготовительной оснастки средствами программного интерфейса и включающий три компонента, состоит в следующем:

- разработка концептуальной модели проектирования средствами объектно-ориентированного программирования;
- создание компонентов в виде элементов проектирования, отвечающих за построение отдельных деталей, входящих в рассматриваемую сборочную единицу;
-создание модели сборочного узла с выгрузкой информации на устройство хранения данных;
-разработка динамически-управляемого интерфейса в среде NX и согласованная работа с вызываемым прикладным модулем.

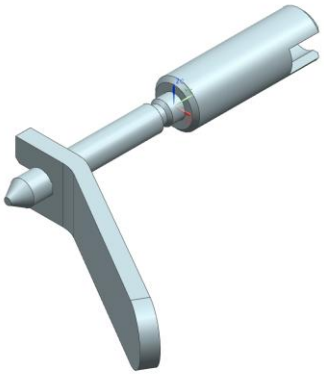

Рис.1. Узел заготовительной оснастки на примере фиксатора

КОНЦЕПТУАЛЬНАЯ МОДЕЛЬ ПРОЕКТИРОВАНИЯ

Проектирование подсистемы моделирования в NX включает в себя реализацию интерфейсной части приложения с включением программной реализации объектов и методов посредством применения NX Open API, а также средствами языка высокого уровня. Описание и разработка интерфейсной части будет рассмотрена позднее.

Объект сборочного узла, который включает в себя метод инициализации общей схемы программного моделирования, члены-методы разработки, построения и выгрузки на носитель информации полученных компонентов сборки фиксатора с набором входящих параметров и основной метод, связывающий компоненты согласно указанным параметрам метода инициализации, представляет собой законченный блок программного приложения. Вызов его позволит произвести построение твердотельной модели сборки с учетом требуемых параметров. Модель реализации представлена на рис. 2, согласно предложенной структурной схеме проектирования.

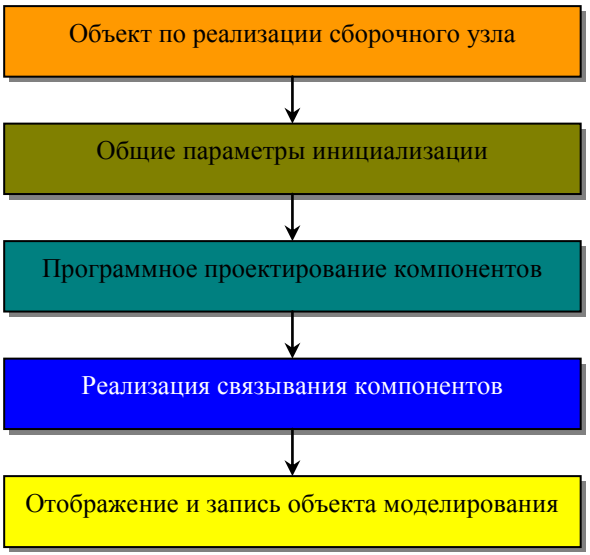

Рис. 2. Схема проектирования прикладного компонента NX

Программную реализацию удобнее представить с учетом концепций объектно-ориентировочного программирования (ОПП) в виде конструкции класса с закрытой и открытой частями методов реализации твердотельных моделей. Структурная схема проектирования приобретет при этом следующий вид (рис. 3).

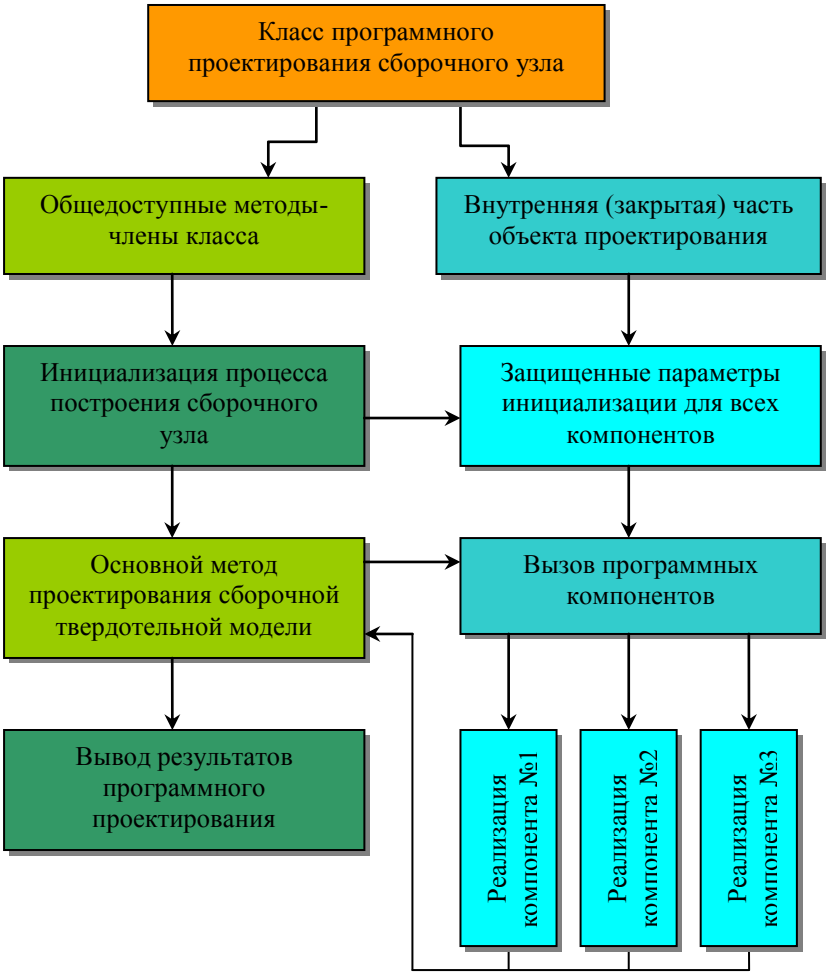

Рис. 3. Структурная реализация процесса построения сборочного узла фиксатора

Реализация на языке программирования C# будет выглядеть согласно листингу, приведенному ниже.

```
//Класс сборки по конструкции фиксатора стыка
    class A_653_0903_1
    {
        //Параметры для всех компонентов
        private double dad, dal, daH;


        //Конструктор класса фиксатора
        public A_653_0903_1 (double x,double y,double z)
        {
            dad = x;
            dal = y;
            daH = z;
        }

//Построение твердотельной модели болта
//Шифр болта 653.0903-X/1
private bool bolt(double dd1, double dd, double dd2,
double dm, double dn, double dl1, double dl2, double
dl3, double df, double dl = 1);

//Построение твердотельной модели гайки
//Шифр гайки 653.0903-X/2
private bool gayka(double dd,double dD,double dH,double
dd1, double dn,double dl,double dl1,double df);

//Построение твердотельной модели чеки
//Шифр чеки 653.0903-X/3
private bool cheka(double dm,double dH,double dh,double
dR, double dn,double dl,double dr);

/Сборочный узел (основной метод класса)
 public bool assembly();

};

//Создание переменной на класс построения элементов
фиксатора и сборочного узла
A_653_0903_1 temp;
temp = new A_653_0903_1(6.4,25,35);
temp.assembly();
```

Класс A_653_0903_1 является основным блоком по реализации и построению сборочной конструкции в NX и связан с некоторой переменной окружения класса-temp. В процессе инициализации класса запущен конструктор с параметрами: диаметром и длинами, определяющими условия проведения сборки.

```
temp = new A_653_0903_1(6.4,25,35);
```

Основной метод проектирования сборочной твердотельной модели может быть запущен по запросу через переменную инициализации с возможностью внесения дополнительных параметров в условие построения. Метод является открытым по архитектуре проектирования.

```
temp.assembly();
```

Реализация компонентов внесена в защищенную часть класса и вызывается исключительно из основного метода. Компоненты содержат наборы параметров, которые определены пользователем на стадии диалогового выбора значений из базы данных.

```
private bool bolt(..);
private bool gayka(..);
private bool cheka(..);
```

Закрытые компоненты представлены в виде функций, методов класса, возвращаемых логическое значение: (TRUE) в случае успешного завершения и записи результатов, (FALSE)- в противном случае.

СОЗДАНИЕ КОМПОНЕНТОВ ПРОЕКТИРОВАНИЯ

Программная реализация компонентов твердотельных моделей сводится к выполнению базовых операций, выполняемых аналогично как при ведении проектов инженером-конструктором. Следуя разработанной схеме реализации,

программные компоненты твердотельных моделей рекомендуется определить в закрытой части класса. Последовательным вызовом созданных блоков с набором параметров решается задача подготовительного этапа сборочного узла.

Для получений компонента твердотельной модели болта (рис. 4) необходимо выполнить операцию вращения спроектированного эскиза, указать кромки и добавить конструктивный элемент-фаску, произвести логическое вычитание эскиза из твердотельной модели для получения проточки.

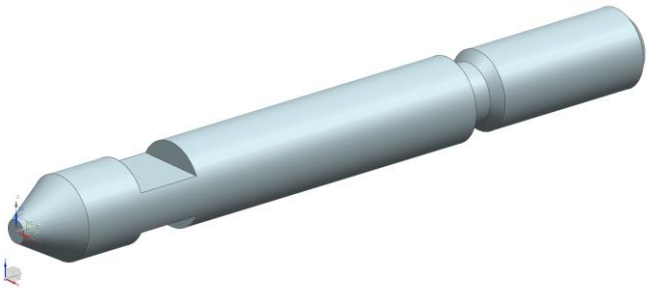

Рис. 4. Компонент твердотельной модели болта

Программная реализация модели болта с учетом передаваемых параметров в функцию выглядит следующим образом:

```
private bool bolt(double dd1, double dd, double dd2,
double dm, double dn, double dl1, double dl2, double
dl3, double df, double dl = 1)
        {
              return flag;
        }
```

Параметры для передачи в функцию представлены размерными характеристиками проектируемой модели: dl, dl_1, dl_2, dl_3,-длины соответствующих участков модели, df-фаска, dm-

высота проточки, dd,dd_1,dd_2- диаметральные составляющие проектируемого компонента. В случае успешного построения программного компонента функция вернет соответствующее логическое значение в переменной flags.

Построение компонента внутри функции связано с рядом функций, определенных в NX Open API. Для инициализации процесса построения необходимо выполнить следующий программный код. Комментарии приведены в листинге программы.

```
//Присвоение глобального параметра
dl = dal;
//Получение сеанса начала работы с объектом
Session theSession = Session.GetSession();
//Создаем новую деталь, объявляя переменную fileNew1
FileNew fileNew1;
fileNew1 = theSession.Parts.FileNew();
//Используем шаблон модели в миллиметрах
fileNew1.TemplateFileName = "model-plain-1-mm-
template.prt";
//Приложение, связанное с процессом моделирования
fileNew1.Application = FileNewApplication.Modeling;
//Создаваемый объект будет выполнен в указанных
//единицах измерения, в данном случае- миллиметрах
fileNew1.Units = NXOpen.Part.Units.Millimeters;
//Путь сохранения файла
fileNew1.NewFileName = "D:\\bolt.prt";
```

Следующий этап программного проектирования указанного компонента заключается в построении контура и вращении его вокруг выбранной оси. Основанием вращения будет выступать одна из сторон спроектированного эскиза. Выбор плоскостей, определение исходных точек построения, системы координат приведены в листинге

```
//Создать новый эскиз на плоскости построения
sketchInPlaceBuilder1 =
workPart.Sketches.CreateNewSketchInPlaceBuilder(nullSket
ch);
//Создать переменную окружения по взятию значений единиц
//измерения
```

```
 Unit unit1 =
(Unit)workPart.UnitCollection.FindObject("MilliMeter");
//Построение эскиза по указанному пути (определение
исходной //плоскости, системы координат)
SketchAlongPathBuilder sketchAlongPathBuilder1;
sketchAlongPathBuilder1 =
workPart.Sketches.CreateSketchAlongPathBuilder(nullSketc
h);
sketchAlongPathBuilder1.PlaneLocation.Expression.RightHa
ndSide = "0";
//Определить координатную плоскость построения
DatumPlane datumPlane1 =
(DatumPlane)workPart.Datums.FindObject("DATUM_CSYS(0) XZ
plane");
//Создать новый точечный объект
Point3d point1 = new Point3d();
//Установить точку на координатной плоскости
sketchInPlaceBuilder1.PlaneOrFace.SetValue(datumPlane1,
workPart.ModelingViews.WorkView, point1);
NXOpen.Features.DatumCsys datumCsys1 =
(NXOpen.Features.DatumCsys)workPart.Features.FindObject(
"DATUM_CSYS(0)");
Point point2 = (Point)datumCsys1.FindObject("HANDLE R-
850");
sketchInPlaceBuilder1.SketchOrigin = point2;
sketchInPlaceBuilder1.PlaneOrFace.Value = null;
sketchInPlaceBuilder1.PlaneOrFace.Value = datumPlane1;
//Определяем ось Y, принадлежащей координатной плоскости
DatumAxis datumAxis1 =
(DatumAxis)workPart.Datums.FindObject("DATUM_CSYS(0) Y
axis");
sketchInPlaceBuilder1.Axis.Value = datumAxis1;
```

Построение замкнутого контура производится следующим с учетом параметров, заданных пользователем.

```
Point3d startPoint1 = new Point3d(0.0, 0.0, 0.0);
Point3d endPoint1 = new Point3d(0.0, 0, dd1 / 2);
//Определить переменную окружения линии
Line line1;
//Создать линию по двум точкам
line1 = workPart.Curves.CreateLine(startPoint1,
endPoint1);
//В активном эскизе добавить линию без привязок
```

```
theSession.ActiveSketch.AddGeometry(line1,
NXOpen.Sketch.InferConstraintsOption.InferNoConstraints)
;
//Определить переменную окружения ограничений в эскизе
NXOpen.Sketch.ConstraintGeometry geom1_1;
//Связать линию с геометрией ограничений
geom1_1.Geometry = line1;
//Ограничение по начальной вершине
geom1_1.PointType =
NXOpen.Sketch.ConstraintPointType.StartVertex;
//Сплайн определяющих позиций точек-от 0.
geom1_1.SplineDefiningPointIndex = 0;
//Определить переменную окружения ограничений в эскизе
NXOpen.Sketch.ConstraintGeometry geom2_1;
//Связать точку с созданной геометрией
geom2_1.Geometry = point2;
geom1_1.SplineDefiningPointIndex = 0;
geom2_1.PointType =
NXOpen.Sketch.ConstraintPointType.None;
geom2_1.SplineDefiningPointIndex = 0;
//Определить переменную окружения ограничений геометрии
эскиза
SketchGeometricConstraint sketchGeometricConstraint1;
//Создать взаимосвязь из двух ограничений в активном
эскизе
sketchGeometricConstraint1 =
theSession.ActiveSketch.CreateCoincidentConstraint(geom1
_1, geom2_1);
//Определить переменную окружения ограничений в эскизе
NXOpen.Sketch.ConstraintGeometry geom1;
//Связать с линией
geom1.Geometry = line1;
geom1.PointType =
NXOpen.Sketch.ConstraintPointType.None;
geom1.SplineDefiningPointIndex = 0;
SketchGeometricConstraint sketchGeometricConstraint2;
sketchGeometricConstraint2 =
theSession.ActiveSketch.CreateVerticalConstraint(geom1);
```

Вращение компонента производится в следующей сессии кода:

```
Section section3;
NXOpen.Features.RevolveBuilder revolveBuilder1;
```

```
revolveBuilder1 =
workPart.Features.CreateRevolveBuilder(nullFeatures_Feat
ure);
//Установка начального положения поворота(в градусах)
revolveBuilder1.Limits.StartExtend.Value.RightHandSide =
"0";
//Установка конечного положения поворота(в градусах)
revolveBuilder1.Limits.EndExtend.Value.RightHandSide =
"360";
revolveBuilder1.Offset.StartOffset.RightHandSide = "0";
revolveBuilder1.Offset.EndOffset.RightHandSide = "5";
section3.SetAllowedEntityTypes(NXOpen.Section.AllowTypes
.OnlyCurves);
NXOpen.Features.Feature[] features1 = new
NXOpen.Features.Feature[1];
NXOpen.Features.SketchFeature sketchFeature1 =
(NXOpen.Features.SketchFeature)feature1;
features1[0] = sketchFeature1;
CurveFeatureRule curveFeatureRule1;
curveFeatureRule1 =
workPart.ScRuleFactory.CreateRuleCurveFeature(features1)
;
section3.AllowSelfIntersection(false);
SelectionIntentRule[] rules4 = new
SelectionIntentRule[1];
rules4[0] = curveFeatureRule1;
Point3d helpPoint1 = new Point3d();
section3.AddToSection(rules4, line2, nullNXObject,
nullNXObject, helpPoint1, NXOpen.Section.Mode.Create,
false);
revolveBuilder1.Section = section3;
Direction direction1;
direction1 = workPart.Directions.CreateDirection(line8,
Sense.Forward,
NXOpen.SmartObject.UpdateOption.WithinModeling);
Point nullPoint = null;
Axis axis1;
axis1 = workPart.Axes.CreateAxis(nullPoint, direction1,
NXOpen.SmartObject.UpdateOption.WithinModeling);
revolveBuilder1.Axis = axis1;
revolveBuilder1.ParentFeatureInternal = false;
NXOpen.Features.Feature feature2;
feature2 = revolveBuilder1.CommitFeature();
```

Построение фасок происходит согласно ниже приведенному листингу программы:

```
chamferBuilder1.Method =
NXOpen.Features.ChamferBuilder.OffsetMethod.EdgesAlongFa
ces;
chamferBuilder1.Option =
NXOpen.Features.ChamferBuilder.ChamferOption.SymmetricOf
fsets;
ScCollector scCollector1;
scCollector1 = workPart.ScCollectors.CreateCollector();
NXOpen.Features.Revolve revolve1 =
(NXOpen.Features.Revolve)feature2;
Edge edge1 = (Edge)revolve1.FindObject("EDGE * [CURVE 2
0] * [CURVE 3 0] {}");
Edge nullEdge = null;
EdgeTangentRule edgeTangentRule1;
edgeTangentRule1 =
workPart.ScRuleFactory.CreateRuleEdgeTangent(edge1,
nullEdge, false, 0.5, false, false);
SelectionIntentRule[] rules5 = new
SelectionIntentRule[1];
rules5[0] = edgeTangentRule1;
scCollector1.ReplaceRules(rules5, false);
chamferBuilder1.SmartCollector = scCollector1;
EdgeTangentRule edgeTangentRule2;
edgeTangentRule2 =
workPart.ScRuleFactory.CreateRuleEdgeTangent(edge1,
nullEdge, false, 0.5, false, false);
            Edge edge2 = (Edge)revolve1.FindObject("EDGE
* [CURVE 3 0] * [CURVE 4 0] {}");
            EdgeTangentRule edgeTangentRule3;
            edgeTangentRule3 =
workPart.ScRuleFactory.CreateRuleEdgeTangent(edge2,
nullEdge, false, 0.5, false, false);
            SelectionIntentRule[] rules6 = new
SelectionIntentRule[2];
            rules6[0] = edgeTangentRule2;
            rules6[1] = edgeTangentRule3;
            scCollector1.ReplaceRules(rules6, false);
            chamferBuilder1.SmartCollector =
scCollector1;
            string str1 = "0.", str2 =
Convert.ToString(((dd - dd2) / 2) * 100);
```

```
            chamferBuilder1.FirstOffsetExp.RightHandSide
= str1 + str2;

chamferBuilder1.SecondOffsetExp.RightHandSide = str1 +
str2;
```

Эскиз для построения лыски (частичное построение элемента) строится методами, аналогичными ранее приведенным способом.

Вторая часть эскиза лыски формируется с помощью команды “зеркальное отражение”, частичный листинг которой приведен далее:

```
sketchMirrorPatternBuilder1 =
workPart.Sketches.CreateSketchMirrorPatternBuilder
(nullSketchPattern);
section6.SetAllowedEntityTypes(NXOpen.Section.AllowTypes
.CurvesAndPoints);

curveFeatureChainRule1 =
workPart.ScRuleFactory.CreateRuleCurveFeatureChain(featu
res2, line12, nullCurve, false, 1e-008);
section6.AddToSection(rules8, line12, nullNXObject,
nullNXObject, helpPoint2, NXOpen.Section.Mode.Create,
false);
DatumAxis datumAxis2 =
(DatumAxis)workPart.Datums.FindObject("SKETCH(5:1B) X
axis");
sketchMirrorPatternBuilder1.DirectionObject.Value =
datumAxis2;
nXObject6 = sketchMirrorPatternBuilder1.Commit();
NXObject[] objects1;
objects1 =
sketchMirrorPatternBuilder1.GetCommittedObjects();
sketchMirrorPatternBuilder1.Destroy();
```

Получение лысок осуществляется с помощью операции выдавливание. Сокращенный листинг операции с краткими комментариями приведен ниже:

```
extrudeBuilder1.BooleanOperation.Type =
NXOpen.GeometricUtilities.BooleanOperation.BooleanType.C
reate;
Body[] targetBodies1 = new Body[1];
Body nullBody = null;
targetBodies1[0] = nullBody;
extrudeBuilder1.BooleanOperation.SetTargetBodies(targetB
odies1);
section7.SetAllowedEntityTypes
(NXOpen.Section.AllowTypes.OnlyCurves);
ICurve[] curves1 = new ICurve[8];
curves1[0] = line9;
Line line13 = (Line)
theSession.ActiveSketch.FindObject("Curve Line13");
curves1[1] = line13;
curves1[2] = line10;
Line line14 =
Point3d seedPoint1 =
new Point3d(0.0, dl2 + dl3 + 2.5, (dd1 - dm) / 2);
RegionBoundaryRule regionBoundaryRule1;
regionBoundaryRule1 =
workPart.ScRuleFactory.CreateRuleRegionBoundary(theSessi
on.ActiveSketch, curves1, seedPoint1, 0.0254);
ICurve[] curves2 = new ICurve[8];
curves2[0] = line9;
RegionBoundaryRule regionBoundaryRule2;
regionBoundaryRule2 =
workPart.ScRuleFactory.CreateRuleRegionBoundary(theSessi
on.ActiveSketch, curves2, seedPoint2, 0.0254);
section7.AddToSection(rules9, nullNXObject,
nullNXObject, nullNXObject, helpPoint3,
NXOpen.Section.Mode.Create, false);
Direction direction2;
direction2 =
workPart.Directions.CreateDirection(theSession.ActiveSke
tch, Sense.Forward,
NXOpen.SmartObject.UpdateOption.WithinModeling);
extrudeBuilder1.Direction = direction2;
extrudeBuilder1.BooleanOperation.SetTargetBodies(targetB
odies2);
extrudeBuilder1.BooleanOperation.Type =
NXOpen.GeometricUtilities.BooleanOperation.BooleanType.U
nite;
```

```
extrudeBuilder1.Limits.StartExtend.TrimType =
NXOpen.GeometricUtilities.Extend.ExtendType.ThroughAll;
DisplayableObject nullDisplayableObject = null;
extrudeBuilder1.Limits.StartExtend.Target =
nullDisplayableObject;
```

По завершению построения детали производится ее сохранение средствами NX:

```
PartSaveStatus partSaveStatus2;
partSaveStatus2 =
workPart.Save(NXOpen.BasePart.SaveComponents.True,
```

Для получений компонента твердотельной модели гайки (рис. 5) необходимо проделать аналогичные действия по проектированию, согласно предыдущему этапу построения.

Гайка формируется при помощи операции вращение, выдавливание и фаска. Исходные данные представлены рядом диаметральных составляющих -dD,dd1; резьбой dd с глубиной dl, шириной и высотой проточки dl1 и dn, а также общим габаритным размером гайки dH. Фаски строятся с соответствующей величиной смещения df с заданным углом в 45°

Алгоритм построения гайки следующий: основной элемент модели получают с помощью операции вращения, далее создаются фаски, после чего осуществляется вырез под ключ.

Программная реализация модели гайки с учетом передаваемых параметров в функцию выглядит следующим образом:

```
private bool gayka(double dd,double dD,double dH,double
dd1, double dn,double dl,double dl1,double df)
        {
              return flag;
        }
```

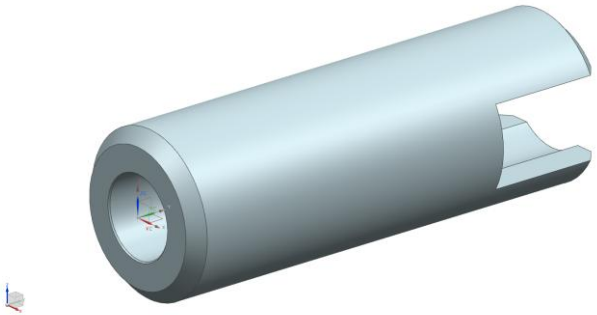

Рис. 5. Компонент твердотельной модели гайки

Листинг функции построения гайки подобен предыдущему листингу функции построения твердотельной модели болта, поэтому программная реализация далее не рассматривается.

Для реализации твердотельной модели детали “чека” (рис. 6) необходимо выполнить следующие операции: построить эскиз, показывающий вид детали сбоку без учета фасок и скруглений, осуществить выдавливание эскиза для получения твердого тела; построить эскиз на передней поверхности построенного тела, который будет использоваться для получения выреза, получить вырез с помощью операции выдавливания, создать фаски на передних ребрах выреза, скруглить внутренние углы выреза, создать скругление торца чеки, противоположного вырезу.

Исходные данные представлены рядом составляющих: dl-общая длина чеки в проекции на плоскость, dm и dh-ширина и длина проточки, dr-радиусы скругления проточки, dH-длина до участка перегиба чеки, dR-краевое скругление чеки.

Алгоритм построения чеки следующий: производится построение замкнутого контура, согласно представленной модели, далее выполняется операция выдавливания, скругление участка перегиба, построение эскиза проточки с последующим логическим вычитанием из имеющегося твердотельного элемента, производится скругление оконечных участков чеки.

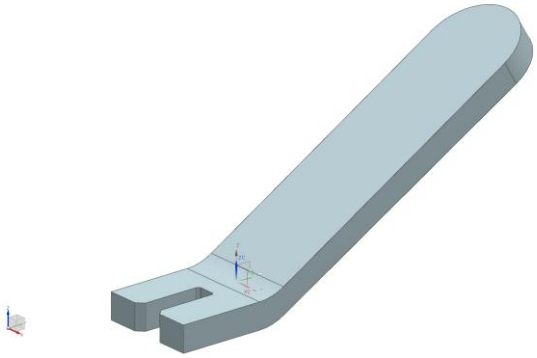

Рис. 6. Компонент твердотельной модели чеки

Программная реализация модели чеки с учетом передаваемых параметров в функцию выглядит следующим образом:

```
private bool cheka(double dm,double dH,double dh,double
dR, double dn,double dl,double dr)
        {
              return flag;
        }
```

Создание чеки с помощью NXOpen API начинается с текста, отвечающего за создание файла:

```
//создание нового файла
FileNew fileNew1;
fileNew1 = theSession.Parts.FileNew();
fileNew1.TemplateFileName = "model-plain-1-mm-
template.prt";
//включение режима моделирования
fileNew1.Application = FileNewApplication.Modeling;
//единица измерения - мм
fileNew1.Units = NXOpen.Part.Units.Millimeters;
//указание имени файла создаваемой детали
fileNew1.NewFileName = "D:\\cheka.prt";
```

Инициализация эскиза осуществляется аналогично деталям, рассмотренным ранее. Эскиз создается в плоскости XZ:

```
//задание плоскости, в которой будет строится эскиз
DatumPlane datumPlane1 =
(DatumPlane)workPart.Datums.FindObject("DATUM_CSYS(0) XZ
plane");
//задание координатной оси
DatumAxis datumAxis1 =
(DatumAxis)workPart.Datums.FindObject("DATUM_CSYS(0) Y
axis");
```

Содержание эскиза создается с помощью отдельных отрезков, координаты которых вычисляются, исходя из параметрической таблицы детали.

Следующий фрагмент отвечает за задание отрезка параметрически:

```
//задание первой точки эскиза базового элемента чеки
через параметры из таблицы, описывающей геометрию чеки
Point3d startPoint1 = new Point3d(0.0, 0.0, 0.0);
//задание второй точки эскиза базового элемента чеки
через параметры из таблицы, описывающей геометрию чеки
Point3d endPoint1 = new Point3d(0.0, -dH, 0);
//задание отрезка, связывающего первую и вторую точку
Line line1;
line1 = workPart.Curves.CreateLine(startPoint1,
endPoint1);
```

Заданные отрезки добавляются в геометрию эскиза с помощью следующей типовой конструкции:

```
//добавление созданного отрезка в геометрию эскиза
theSession.ActiveSketch.AddGeometry(line1,
NXOpen.Sketch.InferConstraintsOption.InferNoConstraints)
;
NXOpen.Sketch.ConstraintGeometry geom1_1;
geom1_1.Geometry = line1;
geom1_1.PointType =
NXOpen.Sketch.ConstraintPointType.StartVertex;
geom1_1.SplineDefiningPointIndex = 0;
NXOpen.Sketch.ConstraintGeometry geom2_1;
geom2_1.Geometry = point2;
geom2_1.PointType =
NXOpen.Sketch.ConstraintPointType.None;
```

```
geom2_1.SplineDefiningPointIndex = 0;
```

Ограничения для объектов геометрии эскиза добавляются с помощью приведенного ниже фрагмента (показан текст для создания ограничения “горизонтально”):

```
//введение ограничений эскиза
SketchGeometricConstraint sketchGeometricConstraint1;
sketchGeometricConstraint1 =
theSession.ActiveSketch.CreateCoincidentConstraint(geom1
_1, geom2_1);
NXOpen.Sketch.ConstraintGeometry geom1;
geom1.Geometry = line1;
geom1.PointType =
NXOpen.Sketch.ConstraintPointType.None;
geom1.SplineDefiningPointIndex = 0;
SketchGeometricConstraint sketchGeometricConstraint2;
sketchGeometricConstraint2 =
theSession.ActiveSketch.CreateHorizontalConstraint(geom1
);
```

В операции “выдавливание” необходимо в качестве базовых параметров определить геометрию эскиза, направление вытягивания, начальную и конечную позицию вытягивания.

Ключевым параметров вытягивания является набор двумерных элементов, который задается с помощью следующей конструкции:

```
//задание примитивов, используемых при выдавливании
SelectionIntentRule[] rules1 = new
SelectionIntentRule[1];
rules1[0] = curveFeatureRule1;
Point3d helpPoint1 = new Point3d();
section3.AddToSection(rules1, line6, nullNXObject,
nullNXObject, helpPoint1, NXOpen.Section.Mode.Create,
false);
```

Направление выдавливания задается с помощью следующего блока программного кода:

```
//направление выдавливания
Direction direction1;
direction1 =
workPart.Directions.CreateDirection(sketch1,
Sense.Forward,
NXOpen.SmartObject.UpdateOption.WithinModeling);
extrudeBuilder1.Direction = direction1;
```

Для задания толщины создаваемого тела необходимо задать начальный и конечный параметры выдавливания. В приведенном далее примере эскиз будет находиться в середине формируемого тела:

```
//параметр начала выдавливания
extrudeBuilder1.Limits.StartExtend.Value.RightHandSide =
text(-dn / 2);
//параметр окончания вытягивания
extrudeBuilder1.Limits.EndExtend.Value.RightHandSide =
text(dn / 2);
```

Вырез чеки формируется аналогично. Эскиз выреза создается на передней поверхности базового элемента чеки. Для определения его расположения используется следующая конструкция:

```
//параметрическое задание переднего ребра чеки
string str1 = "EDGE * 150 * 160 {(" + text(dn / 2) +
"," + text(-dH) + ",0)(0," + text(-dH) + ",0)(" +
text(dn / 2) +
"," + text(-dH) + ",0) EXTRUDE(2)}";
Edge edge1 = (Edge)extrude1.FindObject(str1);
Point point3;
point3 = workPart.Points.CreatePoint(edge1, scalar1,
NXOpen.SmartObject.UpdateOption.WithinModeling);
//центральная точка передней поверхности чеки, на
которой будет формироваться эскиз
Face face1 = (Face)extrude1.FindObject
("FACE 150 {(0," + text(-dH / 2) + ",0) EXTRUDE(2)}");
Point3d point4 = new Point3d();
sketchInPlaceBuilder2.PlaneOrFace.SetValue
(face1, workPart.ModelingViews.WorkView, point4);
```

```
sketchInPlaceBuilder2.SketchOrigin = point3;
sketchInPlaceBuilder2.PlaneOrFace.Value = null;
sketchInPlaceBuilder2.PlaneOrFace.Value = face1;
sketchInPlaceBuilder2.Axis.Value = datumAxis1;
```

Параметры базовых поверхностей и ребер задаются параметрически через соответствующие переменные, функция text используется для преобразования типов данных.

Создание геометрии эскиза выреза чеки аналогично описанным ранее фрагментам.

Операция вытягивания для создания выреза чеки аналогично ранее представленной. Основными отличиями операции являются тип булевой операции (в данном случае вычитание) и глубина вытягивания (в данном случае через все).

```
//параметр вытягивания "через все"
extrudeBuilder2.Limits.EndExtend.TrimType =
NXOpen.GeometricUtilities.Extend.ExtendType.ThroughAll;
//булевая операция вычитание
extrudeBuilder2.BooleanOperation.Type =
NXOpen.GeometricUtilities.BooleanOperation.BooleanType.S
ubtract;
```

Фаски создаются при помощи специализированного инструмента. Далее приведен фрагмент кода, который задает параметры фасок (две стороны и угол):

```
//задание параметров фаски
chamferBuilder1.FirstOffset = "0.5";
chamferBuilder1.SecondOffset = "0.5";
chamferBuilder1.Angle = "45.0";
```

Ребро детали, на котором формируется фаска, задается параметрически через базовые размеры детали:

```
//определение (параметрически) ребра, на котором будет
формироваться фаска
Edge edge2 = (Edge)extrude1.FindObject("EDGE * 160
EXTRUDE(4) 140 {(" + text(-dm / 2) + "," + text(-dH) +
```

```
",-4)(" + text(-dm / 2) + "," + text(-dH) + ",-2)(" +
text(-dm / 2) + "," + text(-dH) + ",0) EXTRUDE(2)}");
```

Далее параметры фаски передаются в соответствующий конструктор, показанный в следующем примере:

```
//параметры выбора входных параметров для задания фаски
Edge nullEdge = null;
EdgeTangentRule edgeTangentRule1;
edgeTangentRule1 =
workPart.ScRuleFactory.CreateRuleEdgeTangent(edge2,
nullEdge, false, 0.5, false, false);
SelectionIntentRule[] rules6 = new
SelectionIntentRule[1];
rules6[0] = edgeTangentRule1;
scCollector1.ReplaceRules(rules6, false);
chamferBuilder1.SmartCollector = scCollector1;
```

Скругления задаются с помощью специализированного инструмента. Основными входными параметрами скруглений являются ребра для скруглений и их радиусы.

Ребра задаются с помощью параметрической конструкции, пример которой показан в описании фасок.

Радиусы задаются следующим образом:

```
//параметры радиуса для скруглений
int csIndex1;
csIndex1 = edgeBlendBuilder1.AddChainset(scCollector2,
"0.5");
int csIndex2;
csIndex2 = edgeBlendBuilder1.AddChainset(scCollector3,
"0.5");
```

По завершению построения чека сохраняется, после чего выгружается из NX. По завершению создания чеки возвращаемое функцией значение становится равно true, что свидетельствует о корректном завершении работы:

```
PartSaveStatus partSaveStatus1;
//сохранение файла
```

```
partSaveStatus1 =
workPart.Save(NXOpen.BasePart.SaveComponents.True,
NXOpen.BasePart.CloseAfterSave.False);
partSaveStatus1.Dispose();
//После построения детали выгрузить все из NX
theSession.Parts.CloseAll(NXOpen.BasePart.CloseModified.
CloseModified, null);
flag = true;
return flag;
```

СОЗДАНИЕ МОДЕЛИ СБОРОЧНОГО УЗЛА

Построение сборочного узла происходит из готовых элементов, которые были получены и сохранены на магнитном носителе. Месторасположение спроектированных компонентов находится в переменной окружения, которую по мере создания сборки необходимо вызывать. Для получения сборочного узла требуется внести имеющиеся компоненты в состав сборки и применить ряд ограничений для каждого компонента. Исключение составит только первый, добавленный в сборку, элемент - он будет являться ключевым в данном контексте построения. Для каждого последующего компонента требуется внесение взаимосвязей с элементом, который уже состоит в составе собираемого узла и там определен.

Программная реализация выглядит следующим образом: подготавливается новый документ к созданию, указывается имя сборочного узла, единицы измерения для проведения построений в геометрическом пространстве NX.

```
//Сборочный узел (основной метод класса)
 public bool assembly()
        {
     bool testflag;

    Session theSession = Session.GetSession();
    FileNew fileNew1;
    fileNew1 = theSession.Parts.FileNew();
    fileNew1.TemplateFileName = "assembly-mm-
template.prt";
```

```
    fileNew1.Application =
    FileNewApplication.Assemblies;
    //Определить единицы измерения
    fileNew1.Units = NXOpen.Part.Units.Millimeters;
    fileNew1.NewFileName = "sborka.prt";
    fileNew1.UseBlankTemplate = false;
    fileNew1.MakeDisplayedPart = true;
    NXObject nXObject1;
    nXObject1 = fileNew1.Commit();
    Part workPart = theSession.Parts.Work;
    Part displayPart = theSession.Parts.Display;
    fileNew1.Destroy();
```

После создания сборочного компонента необходимо добавить в контекст построения имеющуюся на диске твердотельную модель. Добавление компонентов начинается с модели болта, в программной реализации на данный файл имеется ссылка.

```
    //   Добавить компонент
    BasePart basePart1;
    PartLoadStatus partLoadStatus1;
    basePart1 = theSession.Parts.OpenBase("bolt.prt",
out partLoadStatus1);
    partLoadStatus1.Dispose();
    NXObject[] objects1 = new NXObject[0];
    //Создание детали
    Part part1 = (Part)basePart1;
    Point3d basePoint1 = new Point3d(0.0, 0.0, 0.0);
    Matrix3x3 orientation1;
    orientation1.Xx = 1.0;
    orientation1.Xy = 0.0;
    orientation1.Xz = 0.0;
    orientation1.Yx = 0.0;
    orientation1.Yy = 1.0;
    orientation1.Yz = 0.0;
    orientation1.Zx = 0.0;
    orientation1.Zy = 0.0;
    orientation1.Zz = 1.0;
    PartLoadStatus partLoadStatus2;
    NXOpen.Assemblies.Component component1;
    //Добавить компонент в дерево построения
   component1 =
workPart.ComponentAssembly.AddComponent(part1, "MODEL",
```

```
"БОЛТ", basePoint1, orientation1, -1, out
partLoadStatus2);
    partLoadStatus2.Dispose();
```

Аналогичным образом в сборку добавляются оставшиеся компоненты с указанием имен и ссылок месторасположения на магнитном носителе.

```
    BasePart basePart2;
    PartLoadStatus partLoadStatus3;
    basePart2 = theSession.Parts.OpenBase("cheka.prt",
out partLoadStatus3);
    partLoadStatus3.Dispose();
```

После добавления очередного компонента вносятся поочередно взаимосвязи между составляющими элементами компонентов между собой, для взаимосвязей используются грани и ребра компонентов, осевые линии, базовые плоскости и т.д. Пример внесения одного из сопряжений в сборочном узле приводится в следующем листинге функции.

```
   Face face1 =
(Face)component1.FindObject("PROTO#.Features|REVOLVED(2)
|FACE [CURVE 7 0]");
    Line line1;
    line1 = workPart.Lines.CreateFaceAxis(face1,
NXOpen.SmartObject.UpdateOption.AfterModeling);
    NXObject[] objects3 = new NXObject[1];
    objects3[0] = line1;
    int nErrs5;
    nErrs5 =
theSession.UpdateManager.AddToDeleteList(objects3);
    Face face2 =
(Face)component2.FindObject("PROTO#.Features|BLEND(7)|FA
CE [EDGE EXTRUDE(2) 140 EXTRUDE(2) 190] {(0,-
0.0774087947303,0.2831597139278) EXTRUDE(2)}");
    Line line2;
    line2 = workPart.Lines.CreateFaceAxis(face2,
NXOpen.SmartObject.UpdateOption.AfterModeling);
    NXObject[] objects4 = new NXObject[1];
    objects4[0] = line2;
    int nErrs6;
```

```
    nErrs6 =
theSession.UpdateManager.AddToDeleteList(objects4);
    Line line3;
    line3 = workPart.Lines.CreateFaceAxis(face1,
NXOpen.SmartObject.UpdateOption.AfterModeling);
    NXObject[] objects5 = new NXObject[1];
    objects5[0] = line3;
    int nErrs7;
    nErrs7 =
theSession.UpdateManager.AddToDeleteList(objects5);
    Line line4;
    line4 = workPart.Lines.CreateFaceAxis(face1,
NXOpen.SmartObject.UpdateOption.AfterModeling);
    NXOpen.Positioning.Constraint constraint1;
    constraint1 =
componentPositioner1.CreateConstraint();
    NXOpen.Positioning.ComponentConstraint
componentConstraint1 =
(NXOpen.Positioning.ComponentConstraint)constraint1;
    componentConstraint1.ConstraintType =
NXOpen.Positioning.Constraint.Type.Center22;
    Face face3 =
(Face)component2.FindObject("PROTO#.Features|EXTRUDE(4)|
FACE 160 {(1.6,-9.75,-2) EXTRUDE(2)}");
    NXOpen.Positioning.ConstraintReference
constraintReference1;
    constraintReference1 =
componentConstraint1.CreateConstraintReference(component
2, face3, false, false, false);
    Point3d helpPoint1 = new Point3d(1.6, -
11.9801627372757, -2.65989597016017);
    constraintReference1.HelpPoint = helpPoint1;
    Face face4 =
(Face)component2.FindObject("PROTO#.Features|EXTRUDE(4)|
FACE 140 {(-1.6,-9.75,-2) EXTRUDE(2)}");
    NXOpen.Positioning.ConstraintReference
constraintReference2;
    constraintReference2 =
componentConstraint1.CreateConstraintReference(component
2, face4, false, false, false);
    Point3d helpPoint2 = new Point3d(-1.6, -
10.7095760150172, -0.866544712247083);
    constraintReference2.HelpPoint = helpPoint2;
```

```
    Face face5 =
(Face)component1.FindObject("PROTO#.Features|EXTRUDE(6)|
FACE 230 {(0,11.5,-1.5) REVOLVED(2)}");
    NXOpen.Positioning.ConstraintReference
constraintReference3;
    constraintReference3 =
componentConstraint1.CreateConstraintReference(component
1, face5, false, false, false);
    Point3d helpPoint3 = new Point3d(2.17926471936455,
11.5686777202895, -1.5);
    constraintReference3.HelpPoint = helpPoint3;
    constraintReference3.SetFixHint(true);
    Face face6 =
(Face)component1.FindObject("PROTO#.Features|EXTRUDE(6)|
FACE 170 {(0,11.5,1.5) REVOLVED(2)}");
    NXOpen.Positioning.ConstraintReference
constraintReference4;
    constraintReference4 =
componentConstraint1.CreateConstraintReference(component
1, face6, false, false, false);
    Point3d helpPoint4 = new Point3d(-2.42345327789197,
12.8085643406996, 1.5);

constraintReference4.HelpPoint = helpPoint4;
    constraintReference4.SetFixHint(true);
    NXObject[] objects6 = new NXObject[1];
    objects6[0] = line4;
    int nErrs8;
    nErrs8 =
theSession.UpdateManager.AddToDeleteList(objects6);
    componentNetwork1.Solve();
    componentNetwork1.Solve();
    componentNetwork1.ResetDisplay();
    componentNetwork1.ApplyToModel();
    componentPositioner1.ClearNetwork();
    int nErrs9;
    nErrs9 =
theSession.UpdateManager.AddToDeleteList(componentNetwor
k1);

componentPositioner1.DeleteNonPersistentConstraints();
    NXOpen.Assemblies.Arrangement
nullAssemblies_Arrangement = null;
```

```
    componentPositioner1.PrimaryArrangement =
nullAssemblies_Arrangement;
    componentPositioner1.EndAssemblyConstraints();
    NXOpen.Positioning.ComponentPositioner
componentPositioner2;
    componentPositioner2 =
workPart.ComponentAssembly.Positioner;
    componentPositioner2.ClearNetwork();
    componentPositioner2.PrimaryArrangement =
arrangement1;
    componentPositioner2.BeginAssemblyConstraints();
    bool allowInterpartPositioning2;
    allowInterpartPositioning2 =
theSession.Preferences.Assemblies.InterpartPositioning;
    NXOpen.Positioning.Network network2;
    network2 = componentPositioner2.EstablishNetwork();
    NXOpen.Positioning.ComponentNetwork
componentNetwork2 =
(NXOpen.Positioning.ComponentNetwork)network2;

componentNetwork2.MoveObjectsState = true;
    componentNetwork2.DisplayComponent =
nullAssemblies_Component;
    componentNetwork2.NetworkArrangementsMode =
NXOpen.Positioning.ComponentNetwork.ArrangementsMode.Exi
sting;

 componentNetwork2.MoveObjectsState = true;
    componentNetwork2.NetworkArrangementsMode =
NXOpen.Positioning.ComponentNetwork.ArrangementsMode.Exi
sting;
```

После проведения всех действий, связанных с добавлением компонентов и взаимосвязей, сборочный узел примет окончательный вид, представленный на рис. 7. Для ориентации вида построенного объекта (на примере изометрического представления) в функцию сборки необходимо добавить следующий программный код:

```
workPart.ModelingViews.WorkView.Orient(NXOpen.View.Canne
d.Isometric, NXOpen.View.ScaleAdjustment.Fit);
```

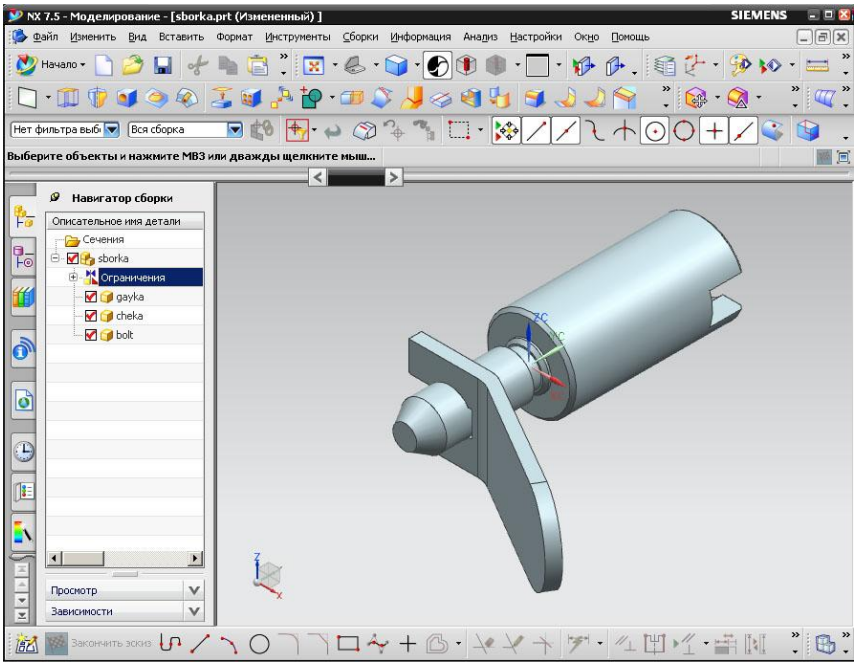

Рис. 7. Сборочный узел, выполненный средствами автоматизации NX

РЕАЛИЗАЦИЯ ИНТЕРАКТИВНОГО ИНТЕРФЕЙСА ПРИКЛАДНОЙ БИБЛИОТЕКИ В СРЕДЕ NX

Рассматриваемый сборочный узел состоит из набора компонентов с управляемыми параметрами. Во время предварительной обработки данных целесообразно внести параметры в каждый компонент. При этом потребуется создание ряда моделей, что не является оптимальным условием для решения поставленной задачи. Для удобства внесения и управления данными для каждого компонента и узла в целом в состав прикладной библиотеки входит база данных, подготовленная средствами Microsoft Access 2007. База данных является основным источником всех параметров, используемых при выборе идентификатора сборки.

Графический интерактивный интерфейс прикладной библиотеки представлен в виде диалогового окна, содержащего

ключевые элементы визуального управления данными. К элементам относятся: схемы выбранных компонентов, полей с фиксированными параметрами, табличным представлением данных на включаемые компоненты болта, чеки и гайка, а также фиксатора в целом. Общий вид интерфейса представлен на рис. 8 и рис. 9.

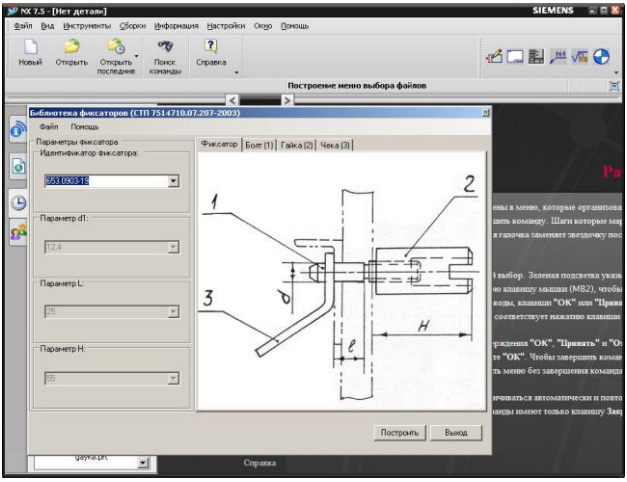

Рис. 8. Интерфейс прикладной библиотеки

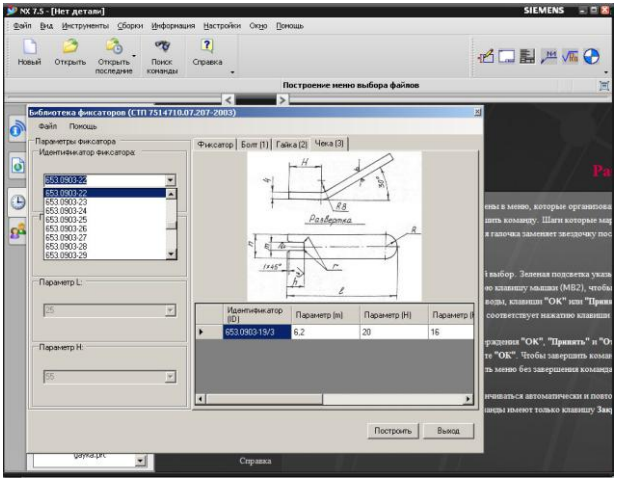

Рис.9. Взаимосвязь между компонентами в режиме назначения параметров из указанных в списке данных

ПОРЯДОК УСТАНОВКИ И ПОДКЛЮЧЕНИЯ ПРИКЛАДНОЙ БИБЛИОТЕКИ В NX

Для удобства подключения библиотеки предлагается вариант с автоматическим размещением требуемых программных компонентов в операционной системе Windows и установленным клиентом NX. Прикладная библиотека разрабатывалась с учетом архитектуры x86 микропроцессоров Intel, а также их аналогов и прошла успешное тестирование в Siemens PLM NX 7.5.0.32 32bit под управлением операционной системы Windows SP3. Возможен запуск прикладной библиотеки в семействе ОС Windows 7 32 битной архитектурой. Для подключения базы данных и ряда компонентов потребуется установка дополнительных модулей визуальной среды программирования .NET, Framework 4 - программной платформы Microsoft и элемент управления базами данных. Для удобства все выше перечисленные компоненты (за исключением платформонезависимой виртуальной машины Framework 4) включены в состав установочного дистрибутива. Установка указанных компонентов для корректной работы **обязательна**! Порядок установки прикладной библиотеки сборочного узла показан на рис. 10. Для вызова библиотеки в среде NX необходимо воспользоваться комбинацией клавиш CTRL+U или соответствующим разделом в меню системы проектирования.

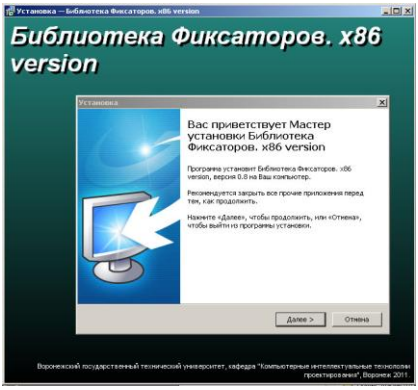

Рис. 10. Установочный модуль прикладной библиотеки

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Интерактивное руководство в системе NX7.5 English по NX Open API.
2. NX для конструктора-машиностроителя / П.С. Гончаров, М.Ю. Ельцов, С.Б. Коршиков, И.В. Лаптев, В.А. Осиюк. -М.: ДМК-Пресс, 2010. - 504 с.
3. Краснов М. Unigraphics для профессионалов / М. Краснов, Ю. Чигишев.- М.: Лори, 2004. – 141 с.
4. Троелсен Э. Язык программирования C# 2010 и платформа .NET 4.0. / Э. Троелсен. 5-е изд. -М.: ООО “И.Д. Вильямс”, 2011. - 1392 с.
5. Шилдт Г. C# 4.0: полное руководство 2010 / Г. Шилдт -М.: ООО “И.Д. Вильямс”, 2011 - 1056 с.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к практическим работам по дисциплине «Разработка САПР»
для студентов направления 230100.64 профиля
«Системы автоматизированного проектирования
в машиностроении» очной формы обучения

Составители
Юров Алексей Николаевич
Паринов Максим Викторович
Чижов Михаил Иванович
Рыжков Владимир Анатольевич

В авторской редакции

Компьютерный набор А.Н. Юрова

Подписано в изданию 10.11.2011.
Уч.-изд. л. 2,1. «С»

ФГБОУВПО «Воронежский государственный технический
университет»
394026 Воронеж, Московский просп., 14