Федеральное агентство по образованию

ГОУ ВПО "Тульский государственный университет"

Технологический факультет

Кафедра "Автоматизированные станочные системы"

Курс *"Разработка САПР"*

Создание САПР на базе КОМПАС 3D

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
ДЛЯ ВЫПОЛНЕНИЯ ЛАБОРАТОРНЫХ РАБОТ
СТУДЕНТАМИ СПЕЦИАЛЬНОСТИ 230104 САПР

ТУЛА 2007

Разработал к.т.н., доц. Троицкий Д.И.

Рассмотрено на заседании кафедры АСС

Зав. кафедрой АСС д.т.н., проф.

_____ Иноземцев А.Н.

1. Основы разработки специализированных САПР

Большинство применяемых в промышленности трехмерных САПР могут быть использованы как основа для построения специализированной САПР, решающей задачу расчета и проектирования конкретного класса изделий. При этом необходимо объединить расчетный модуль, определяющий размерные и иные параметры проектируемого объекта с уже имеющимся в САПР трехмерным геометрическим ядром (

Рис. 1.1).

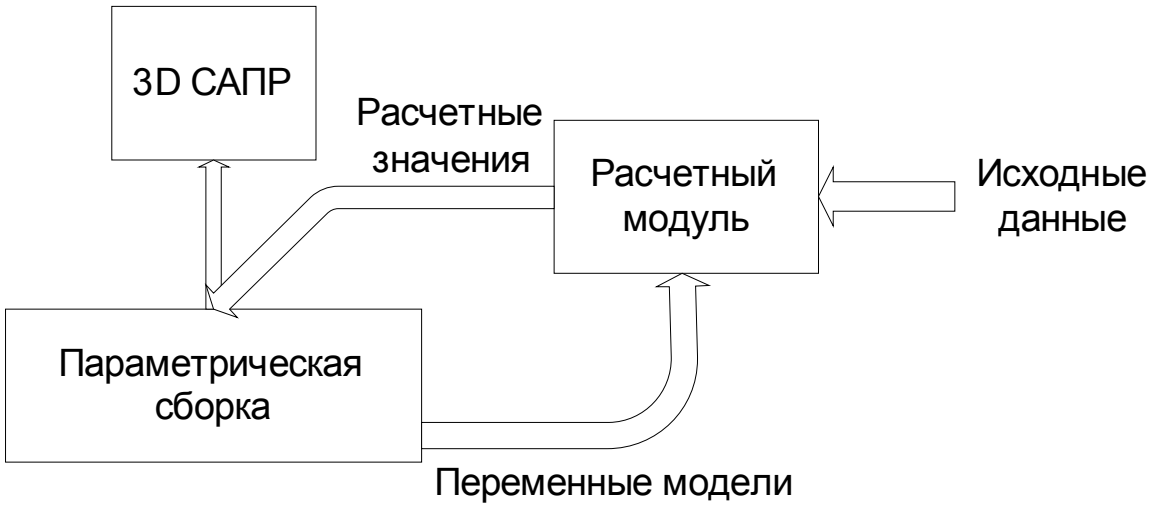

Рис. 1.1 – Структура специализированной САПР.

Для этого сначала создается *параметрическая сборка* проектируемого механизма, в которой ряд размеров вынесен в *переменные модели*. Расчетный модуль (это внешний exe-файл или подключаемая к САПР dll-библиотека, написанные, например, на Delphi) может рассчитать требуемые значения переменных модели и автоматически изменить их, в результате чего будет получен новый вариант 3D сборки. Таким образом, сразу же после расчета будет получена новая геометрия изделия. Разумеется, такой способ накладывает ограничения на функциональность специализированной САПР: можно только менять размеры, но не добавлять или удалять детали и/или их конструктивные элементы (скажем, не удастся сделать модель зубчатого колеса с произвольным числом зубьев). С другой стороны, в большинстве случаев работа конструктора как раз и сводится к модификации ранее созданной геометрии узла в соответствии с новыми расчетными данными, и здесь описываемая специализированная САПР полностью выполняет задачу автоматизации конструкторского труда, выполняя и расчет, и построение модели.

Очевидно, главную сложность представляет не столько выполнение расчетов, сколько организация взаимодействия расчетного модуля и САПР. Исторически сложилось, что большинство современных САПР не поддерживают COM-технологию, что дополнительно затрудняет управление ими из внешней программы. Как правило, такое управление осуществляется при помощи технологии API (Application Programming Interface). API-технология предоставляет программисту набор процедур и функций для управления САПР, но не дает прямого доступа к свойствам и методам объектов внутри САПР, что делает код программы несколько более громоздким и менее понятным.

Мы рассмотрим основы работы с API-интерфейсом САПР КОМПАС 3D версии от 6 и выше. Для использования API-интерфейса из Delphi необходимо прежде всего обзавестись файлами, хранящими прототипы (заголовки) процедур и функций API. Эти файлы имеют названия `ksAuto.pas`, `ksTLB.pas`, `LDefin2D.pas`, `LDefin3D.pas`. Они входят в стандартную поставку КОМПАС 3D и по умолчанию расположены в папке `Program`

Files\Ascon\KOMPAS 3D\SDK\Include.

2. Создание параметрической сборки

Рассмотрим создание специализированной САПР на примере приложения, которое будет рассчитывать параметры простейшего гидроцилиндра и строить его 3D модель (сборку из двух деталей "Цилиндр" и "Поршень").

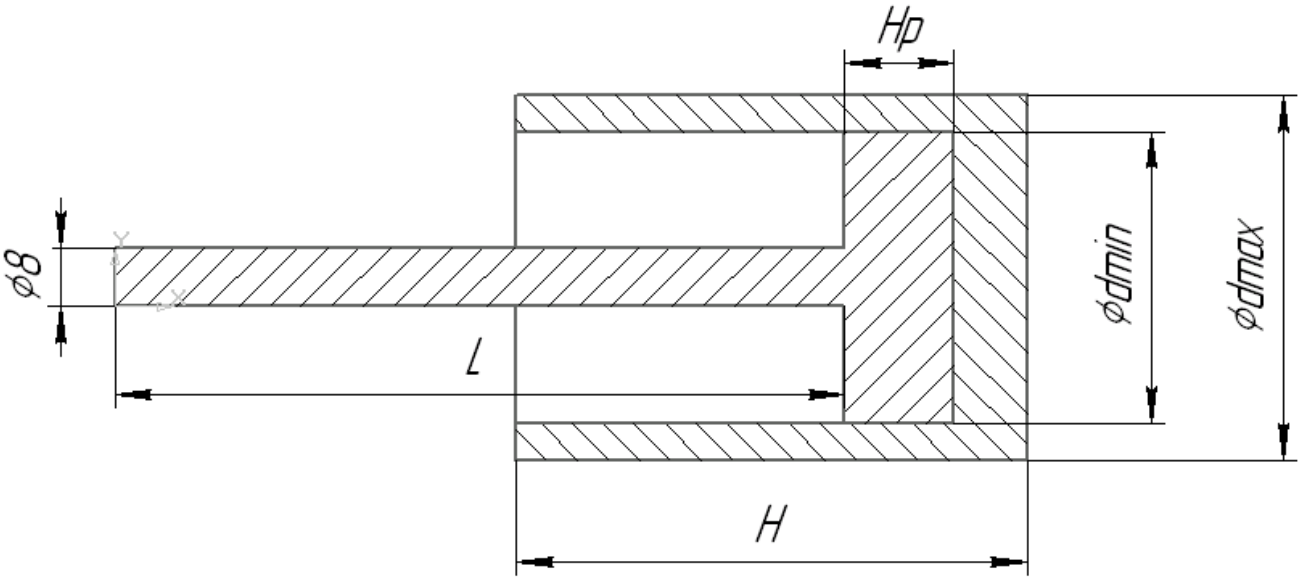

Рис. 2.1 – Проектируемый гидроцилиндр.

Диаметр штока поршня фиксирован и равен 8 мм. Остальные параметры являются переменными и зависят от результатов расчета:

- длина штока *L*;
- диаметр поршня *dmin*;
- наружный диаметр цилиндра *dmax*;
- длина цилиндра *H*;
- высота поршня *Hp*.

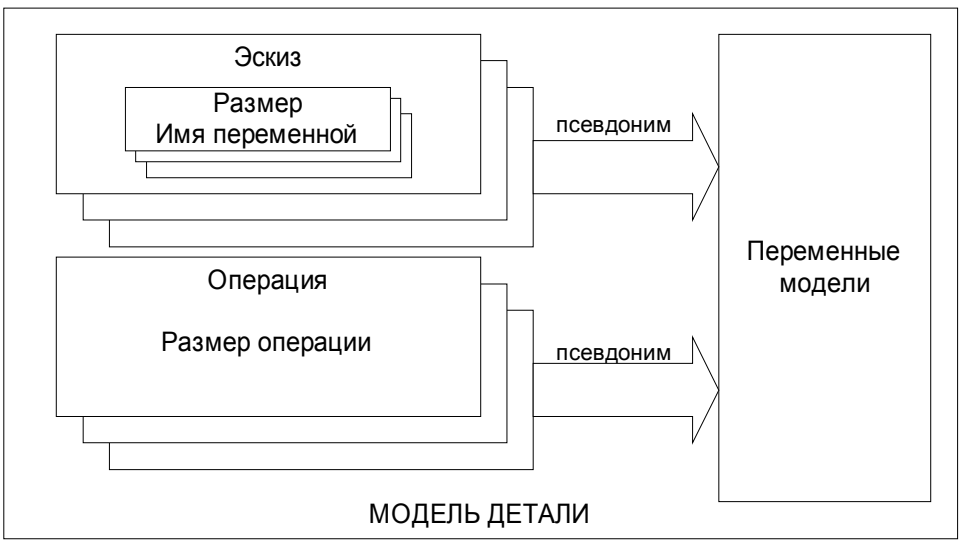

Рис. 2.2 – Структура переменных модели.

Для того, чтобы указанные параметры можно было задавать из внешней программы, их надо объявить как *переменные модели*. В КОМПАС 3D переменными модели могут быть любые размеры, проставленные на эскизах, а также размеры, вводимые при выполнении формообразующих операций (например, высота выдавливания эскиза). Для доступа к переменным на уровне детали их надо объявить как *внешние переменные,* назначив им *псевдонимы*. Псевдоним - это имя, под которым переменная эскиза или размер операции виден на уровне детали (

Рис. 2.2).

2.1. Параметрическая модель с переменными

Построим параметрическую модель детали "Цилиндр". Она полностью определяется тремя переменными: *dmin*, *dmax*, *H* (Рис. 2.1). Проще всего построить полый цилиндр, а затем приклеить к нему сплошное днище. Эскиз полого цилиндра будет иметь вид двух концентрических окружностей. При их образмеривании следует вводить не только номинальные значения размеров (пока произвольные), но и имена переменных (Рис. 2.3).

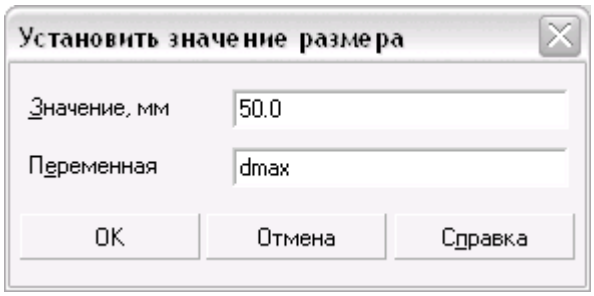

Рис. 2.3 – Окно ввода значения размера имени переменной.

Тогда весь эскиз будет иметь вид (Рис. 2.4):

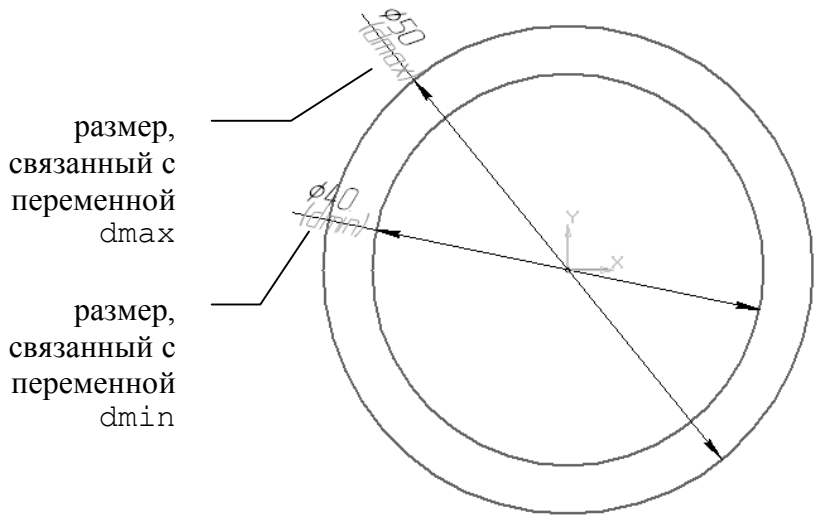

Рис. 2.4 – Эскиз с переменными.

Не выходя из эскиза, нажмите кнопку "Уравнения" f(x). Откроется окон со списком переменных эскиза (Рис. 2.5). У обеих переменных следует поставить галочку "Внешняя". В поле "Комментарий" можно записать произвольный текст (например, "Внешний диаметр цилиндра").

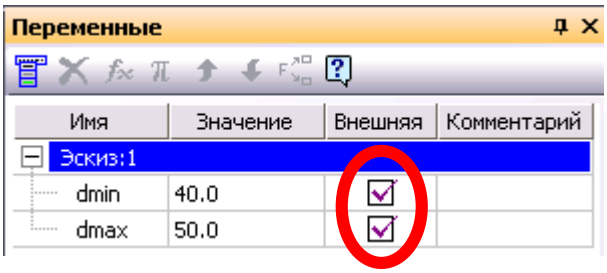

Рис. 2.5 – Окно переменных эскиза.

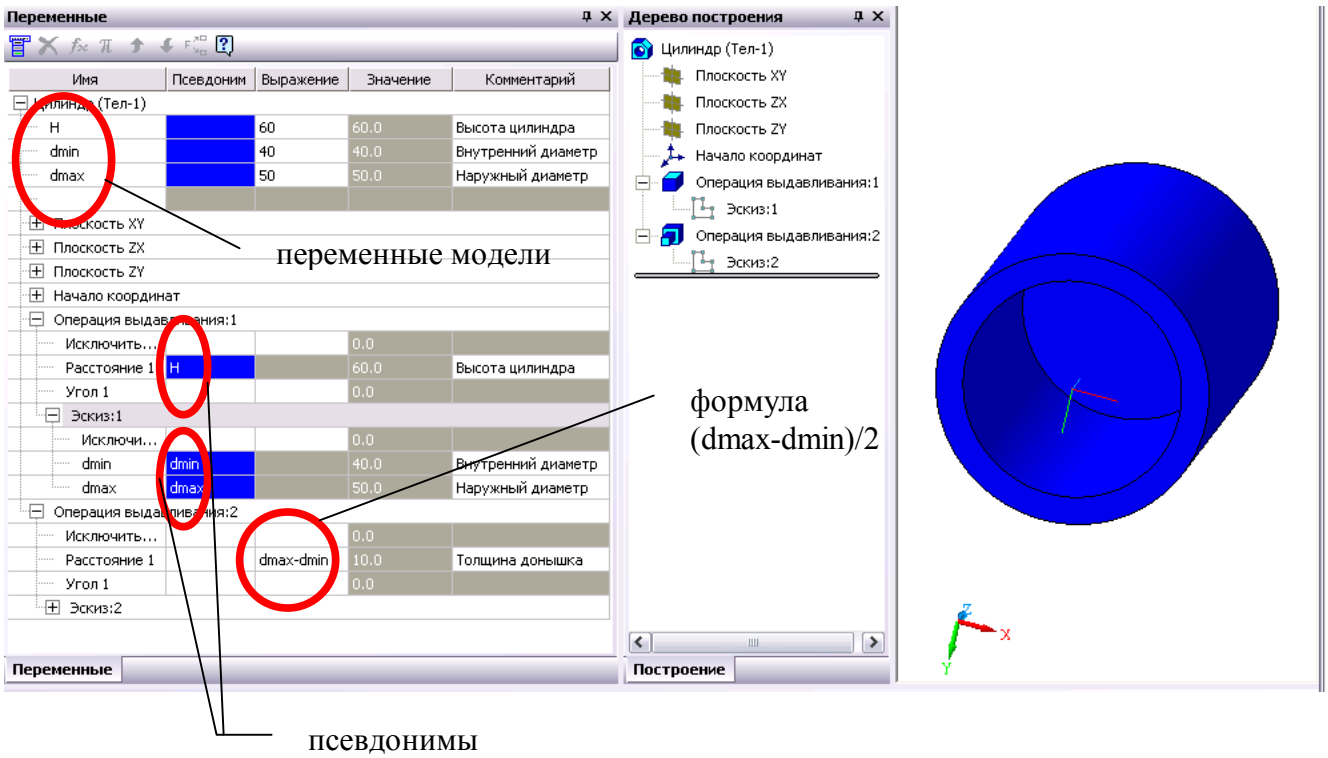

Рис. 2.6 – Переменные модели.

Полученный эскиз следует выдавить на произвольную высоту, получив полый цилиндр. Теперь будем строить днище. Очевидно, что его толщина должна быть равна толщине стенки цилиндра, т.е. *(dmin-dmax)/2*. КОМПАС 3D позволяет задавать уравнения, связывающие переменные модели. Поэтому пока просто построим окружность на одном из торцов цилиндра и выдавим ее на произвольное расстояние. Снова вернемся к окну переменных. Теперь, если нет редактируемого эскиза, в нем отображаются все переменные модели (Рис. 2.6).

В верхней части дерева переменных (ветка "Цилиндр" – по названию детали) идет список переменных модели. Чтобы наши переменные там оказались, надо ввести произвольные псевдонимы для переменных первого эскиза `dmin`, `dmax` и первой операции выдавливания (переменная *H*). Просто введите эти названия в поле "Псевдоним". Для задания величины выдавливания второго эскиза (толщины дна) в поле выражение строки "Расстояние 1" в ветке "Операция выдавливания 2" нужно ввести очевидную формулу `(dmax-dmin)/2`. Теперь, вводя новые численные значения в столбце "Выражение" переменных модели, можно получать различную геометрию цилиндра.

ЗАМЕЧАНИЕ. После ввода новых значений необходимо принудительно перестроить модель, нажав клавишу **F5**.

Все? Еще нет. Необходимо сделать переменные модели доступными на следующем уровне – на уровне сборки. Для этого их надо объявить внешними (точно так же, как внешними объявлялись переменные эскиза для доступа к ним на уровне детали). Щелкните правой

кнопкой мыши по каждой из переменных модели и в контекстном меню отметьте пункт "Внешняя" (Рис. 2.7). Сохраните деталь.

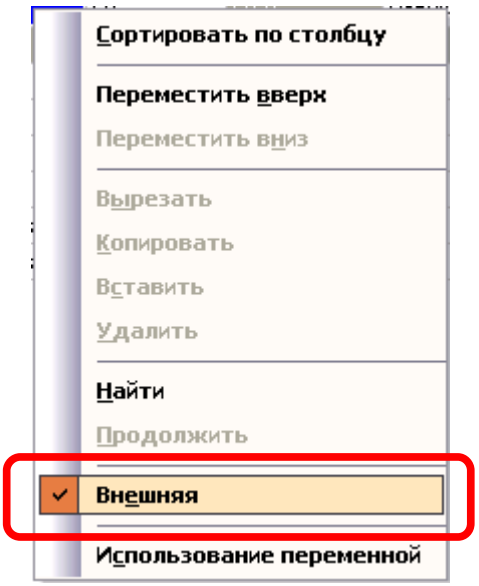

Рис. 2.7 – Контекстное меню переменной модели.

Аналогичным образом постройте поршень, создав в нем внешние переменные модели `Hp` и `L`. Кроме того, необходимо создать переменную `dmin1`, соответствующую наружному диаметру поршня. Она будет рассчитываться в зависимости от значения `dmin` (внутренний диаметр цилиндра) и требуемой величины зазора. Важно, что именно `dmin1` зависит то `dmin`, а не наоборот. Почему? Потому что гораздо проще при изготовлении варьировать диаметр поршня (он изготавливается точением), чем внутренний диаметр цилиндра (он растачивается мерным инструментом).

2.2. Собираем все вместе

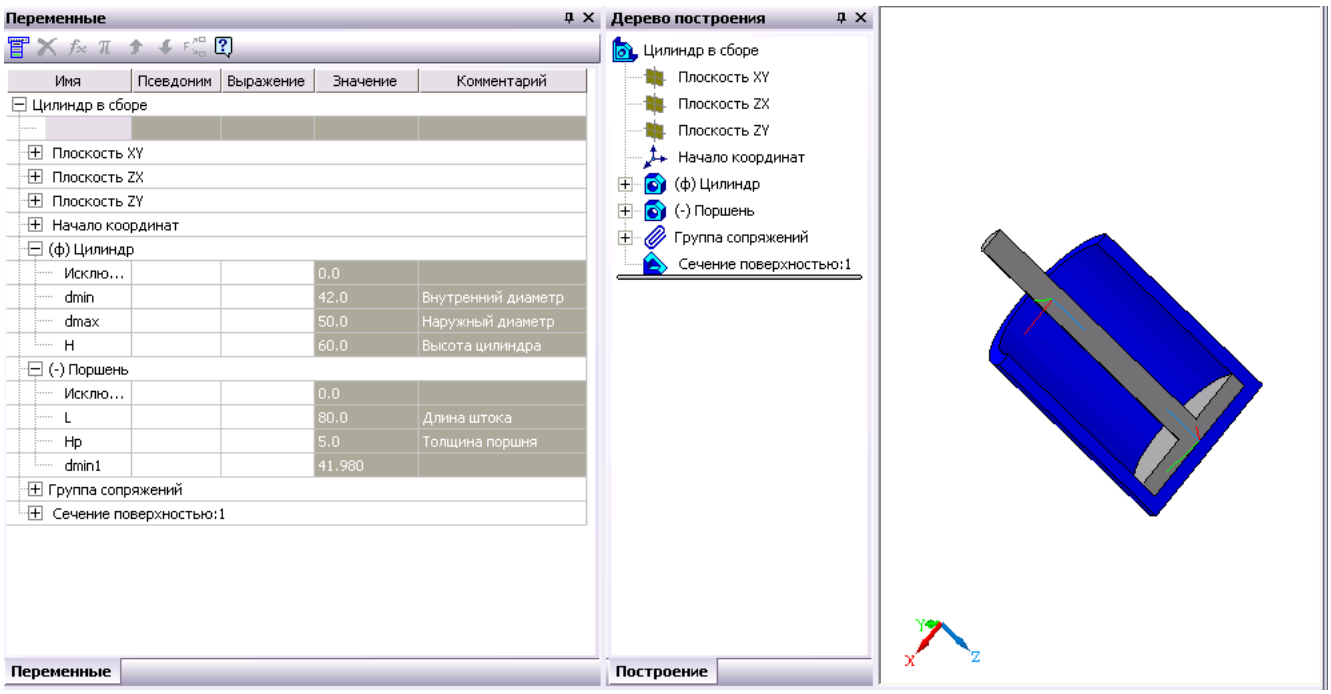

Рис. 2.8 – Окно переменных сборки.

Создадим сборку из деталей "Цилиндр" и "Поршень", наложив на них очевидные сопряжения "Соосность" и "Совпадение" (поршень полностью вдвинут в цилиндр). В окне переменных отображаются все шесть параметров модели: dmax, dmin, dmin1, L, H, Hp. Подготовка параметрической сборки к работе с расчетным модулем на этом завершена.

3. API-интерфейс КОМПАС 3D

Расчетный модуль помимо собственно расчетов должен выполнять следующие функции:

- подключение к КОМПАС 3D и загрузка в него параметрической сборки;
- получение текущих значений переменных деталей, входящих в сборку, названий деталей и названия самой сборки;
- изменение значений переменных, перестроение и сохранение модели.

3.3. Подключение к КОМПАС

Любой проект, работающий с API КОМПАСа, должен содержать в своем составе модули ksAuto.pas, ksTLB.pas, LDefin2D.pas, LDefin3D.pas. Создайте в Delphi новое приложение, выполните команду меню Project → Add to project и добавьте указанные файлы в проект (можно для надежности скопировать их в ту же папку, где находится весь проект). В оператор USES головного модуля проекта добавьте модули ComObj, OleCtrls, ksTLB, ComCtrls.

Для установки связи с КОМПАС введем следующие глобальные переменные:

```
VAR
  kompas:KompasObject; // ссылка на API-объект КОМПАС
  Doc:ksDocument3D; // ссылка на текущий документ КОМПАСа
  KompasHandle:THandle; // ссылка на окно программы КОМПАСа
```

Типы данных KompasObject и ksDocument3D описаны в ранее подключенном модуле ksTLB.

Установление связи с КОМПАС и загрузка в него указанного файла выполняет приведенная ниже функция StartKompas. Она возвращает False, если установить связь с КОМПАС не удалось (скажем, он не установлен на компьютере) и True в случае удачи.

```
function StartKompas(filename:string):boolean;

const ka='Kompas.Application.5';

begin
// подключение к КОМПАС 3D
 Result:=true;
 try
 kompas:=KompasObject(GetActiveOleObject(ka)); // если уже запущен
 except
   try
    kompas:=KompasObject(CreateOleObject(ka)); // если не запущен
   except
    result:=false;
    exit
   end
  end;
  // получение ссылки на окно КОМПАС
```

```
  KompasHandle := kompas.ksGetHWindow;
  // делаем окно КОМПАСа видимым
  kompas.Visible:=true;
  // получение ссылки на текущий документ КОМПАСа
  Doc := ksDocument3D(kompas.ActiveDocument3D);
  // если такой документ есть...
  if Assigned(Doc) then
   // то закрываем его
   Doc.close;
  // создаем новый документ...
  Doc := ksDocument3D(kompas.Document3D);
  // и загружаем в него сборку с именем filename
  Doc.Open(Trim(filename), False);
  // активируем API
  kompas.ActivateControllerAPI
end;
```

Можно вызывать данную функцию, например, следующим образом (на форме есть компонент `LabeledEdit1`, в который пользователь вводит имя файла сборки):

```
if not(StartKompas(Trim(LabeledEdit1.Text))) then
  begin
   messagedlg('Ошибка подключения к КОМПАС',mtError,[mbOK],0);
   exit
  end;
```

В случае успешного выполнения КОМПАС будет запущен, его окно станет видимым и в него будет загружен указанный файл.

3.4. Получение названий деталей в сборке и значений переменных модели

Следующая процедура считывает в массив текстовых строк типа `TStringList` имена деталей в текущей сборке. Под нулевым индексом в массив помещается имя самой сборки:

```
procedure ReadParts(s:TStringList);

var i,num:word;
    parts:ksPartCollection;
    part:ksPart;
begin
  // получение ссылки на список деталей
  parts:=ksPartCollection(doc.PartCollection(true));
  // число деталей
  num:=parts.GetCount;
  s.Clear;
  // деталь с номером -1 есть сама сборка
  s.Add(ksPart(doc.GetPart(-1)).name);
  // цикл по деталям
  for i:=0 to num-1 do
   begin
    // получение ссылки на деталь номер i
    part:=ksPart(parts.GetByIndex(i));
```

```
     // помещаем имя детали в список
    s.Add(part.name);
   end
end;
```

Для использования этой процедуры следует вручную создать объект типа TStringList, например:

```
VAR c:TStringList;
    i:WORD;

BEGIN
 c:=TStringList.Create;
 ReadParts(c);
 Label1.Caption:='Сборка '+c[0];
 FOR i:=1 TO c.Count-1 DO
  Memo1.Lines.Add('Деталь №'+IntToStr(i)+' '+c[i]);
 …
c.Free
…
```

Зная имя детали, можно получить имена ее переменных и их текущие значения. Введем тип данных (запись) для хранения имени переменной и ее значения и динамический массив из таких записей:

```
TYPE TPartVar=RECORD
      VarName:STRING;  // имя переменной
      VarValue:REAL;   // значение переменной
      VarNote:STRING;  // комментарий к переменной
     END;
     TPartVars=ARRAY OF TPartVar;
```

Следующая функция возвращает динамический массив записей с названиями и значениями переменных детали с именем partname:

```
function GetPartVars(partname:STRING):TPartVars;

var vr:ksVariableCollection;
    parts:ksPartCollection;
    part:ksPart;
    univar:ksVariable;
    top,cur,vrr:TTreeNode;
    j, numpart:WORD;

begin
  parts:=ksPartCollection(doc.PartCollection(true));
  // ссылка на деталь с именем partname
  part:=ksPart(parts.GetByName(partname,True,True));
  // ссылка на список переменных детали
  vr:=ksVariableCollection(part.VariableCollection);
  // цикл по переменным детали
  numpart:=vr.GetCount;
  SetLength(result,numpart);
```

```
  for j:=0 to numpart-1 do
     begin
      // ссылка на отдельную переменную
      univar:=ksVariable(vr.GetByIndex(j));
      with result[j] do
       begin
        VarName:=univar.name;
        VarNote:=univar.note;
        VarValue:=univar.value
       end
     end
end;
```

Пример вызова:

```
VAR c:TStringList;
    t:TPartVars;
    i:WORD;

BEGIN
 c:=TStringList.Create;
 ReadParts(c);
 Label1.Caption:='Сборка '+c[0];
 // переменные первой детали
 t:=GetPartVars(c[1])
 FOR i:=0 TO Length(t)-1 DO
  Memo1.Lines.Add(t[i].VarName+'='+FloatToStr(t[i].VarValue));
 …
c.Free
…
```

3.5. Изменение значений переменных и перестроение модели

Следующая процедура заносит вещественное значение `value_` в переменную с именем `varname` детали с именем `partname`:

```
PROCEDURE ChangeVar(partname, varname: STRING; value_:REAL);

VAR vr:ksVariableCollection;
    parts:ksPartCollection;
    part:ksPart;
    vvv:ksVariable;

BEGIN
 // Список деталей
 parts:=ksPartCollection(doc.PartCollection(true));
 // Ищем деталь по имени
 part:=kspart(parts.GetByName(partname,true,true));
 // Список переменных детали
 vr:=ksVariableCollection(part.VariableCollection);
 // Ищем переменную по имени
 vvv:=ksVariable(vr.GetByName(varname,true,true));
```

```
 // Начинаем редактировать деталь
 part.BeginEdit;
 // Меняем значение переменной
 vvv.value:=value_;
 // Обновляем модель
 part.Update;
 part.RebuildModel;
 // Завершаем редактирование детали с сохранением изменений
 part.EndEdit(true);
 // Обновляем сборку
 parts.refresh
END;
```

Пример вызова:

```
VAR c:TStringList;
    t:TPartVars;
    i:WORD;

BEGIN
 c:=TStringList.Create;
 ReadParts(c);
 Label1.Caption:='Сборка '+c[0];
 // переменные первой детали
 t:=GetPartVars(c[1])
 // занести значение 28.5 в первую переменную первой детали
 ChangeVar(c[1],t[0],28.5)
…
c.Free
```

Для сохранения всей сборки после внесения в нее изменений следует вызвать процедуру `Doc.Save`.

4. Снова собираем все вместе

Итак, у нас имеются все необходимые процедуры для работы с КОМПАС. Разумно вынести их в отдельный модуль (например, с именем `KompasAPI.pas`), так как они понадобятся при разработке множества разных специализированных САПР. Для удобства в этом же модуле введены массивы `prt` и `partvar`, чтобы их не приходилось каждый раз описывать в основной программе:

```
unit KompasAPI;

interface

uses ComObj, OleCtrls, ksTLB, Classes, SysUtils;

TYPE TPartVar=RECORD
      VarName:STRING;
      VarNote:STRING;
      VarValue:REAL
     END;
     TPartVars=ARRAY OF TPartVar;
```

```
VAR
  kompas:KompasObject;
  Doc:ksDocument3D;
  KompasHandle:THandle;
  prt:TStringList; // список имен деталей
  partvar:TPartVars;

function StartKompas(filename:string):boolean;
procedure ReadParts(s:TStringList);
function GetPartVars(partname:STRING):TPartVars;
PROCEDURE ChangeVar(partname, varname: STRING; value_:REAL);

implementation
… // вышеописанные процедуры и функции

// раздел инициализации
// выполняется автоматически при запуске программы

initialization

begin
 // выделение памяти под массивы
 prt:=TStringList.Create;
 SetLength(partvar,0)
end;

// раздел финализации
// выполняется автоматически при завершении программы

finalization

begin
 // освобождение памяти
 finalize(partvar);
 prt.Free
end;

end.
```

Готово! Теперь мы имеем мощный инструмент для создания САПР на базе КОМПАС.

5. Универсальный редактор параметрических сборок

Рассмотрим применение вышеописанных процедур и функций для создания универсальной программы, позволяющей редактировать любую сборку, в деталях которой заданы переменные модели. Структуру сборки вместе с переменными удобнее всего выводить в виде дерева (Рис. 5.1).

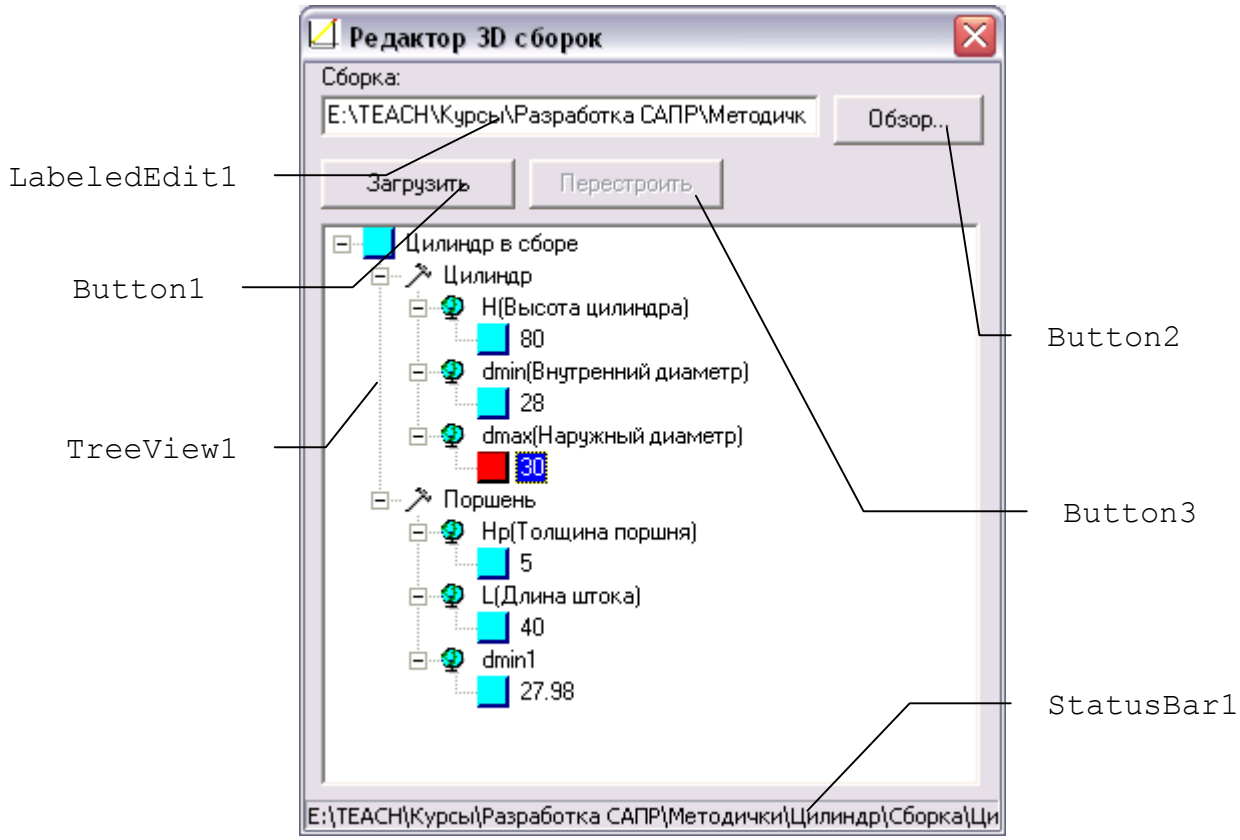

Рис. 5.1 – Окно редактора сборок

Кнопка "Обзор…" вызывает стандартный диалог открытия файла, имя файла при этом заносится в свойство `Text` компонента `LabeledEdit1`:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
 WITH OpenDialog1 DO
  IF Execute THEN
   LabeledEdit1.Text:=FileName
end;
```

Кнопка "Загрузить" (`button1`) устанавливает связь с КОМПАС и выполняет загрузку дерева переменных сборки. Данная кнопка становится активной только при указании правильного пути и имени файла в компоненте `LabeledEdit1`:

```
procedure TForm1.LabeledEdit1Change(Sender: TObject);
begin
 button1.Enabled:=FileExists(Trim(labelededit1.Text))
end;
```

А вот что происходит при ее нажатии:

```
procedure TForm1.Button1Click(Sender: TObject);

begin
 if not(StartKompas(Trim(LabeledEdit1.Text))) then
  begin
   messagedlg('Ошибка подключения к КОМПАС',mtError,[mbOK],0);
```

```
   exit
  end;
  // вывод имени файла в статусную строку
  statusbar1.SimpleText:=Trim(LabeledEdit1.Text);
  ReadParts(prt);
  LoadVars
end;
```

Процедура `LoadVars` создает дерево при помощи компонента `TTreeView`:

```
procedure LoadVars;

var top,vrr,vrr1,vrr2:TTreeNode;
    i,j:word;

begin
  form1.TreeView1.Items.Clear;
  // первый узел дерева – название сборки
  top:=form1.TreeView1.Items.AddChild
  (nil,ksPart(doc.GetPart(-1)).name);
  top.ImageIndex:=0;
  // цикл по деталям
  for i:=1 to prt.Count-1 do
   begin
    // список переменных детали
    partvar:=GetPartVars(prt[i]);
    vrr:=form1.TreeView1.Items.AddChild(top,prt[i]);
    vrr.ImageIndex:=1;
    // цикл по переменным детали
    for j:=0 to length(partvar)-1 do
     begin
      vrr1:=form1.TreeView1.Items.AddChild(vrr,partvar[j].VarName);
      // есть ли комментарий к переменной?
      if partvar[j].VarNote<>'' then
       vrr1.Text:=vrr1.Text+'('+partvar[j].VarNote+')';
      vrr1.ImageIndex:=2;
      vrr2:=form1.TreeView1.Items.AddChild
      (vrr1,FloatToStr(partvar[j].VarValue));
      vrr2.ImageIndex:=3;
      vrr2.SelectedIndex:=4
     end;
    end;
    form1.TreeView1.FullExpand
end;
```

Для отображения разных значков у сборки, детали и переменной к компоненту `TreeVew1` подключен набор рисунков `ImageList1`. Для каждого узла дерева при его создании указывается номер картинки, отображаемой, когда узел не выделен (свойство `ImageIndex`) и картинки, отображаемой при выделении узла (свойство `SelectedIndex`).

Разумно запретить выделение в дереве любых узлов, кроме значений переменных. Очевидно, все значения переменных являются узлами третьего уровня (нулевой уровень – имя сборки, первый – имя детали, второй – имя переменой). На событии `OnChanging` компонента

TreeView1, вызываемом при попытке перехода по узлам дерева, будем выполнять следующий обработчик:

```
procedure TForm1.TreeView1Changing(Sender: TObject; Node: TTreeNode;
  var AllowChange: Boolean);
begin
 AllowChange:=Node.Level=3
end;
```

Данный код разрешает выделять в дереве только узлы третьего уровня.

Отдельно следует решить вопрос о редактировании значений переменных. Перед их занесением в дерево нужно выполнить элементарную проверку на соответствие введенного значения вещественному типу данных. На событие OnEditing компонента TreeView1 поставим следующий обработчик:

```
procedure TForm1.TreeView1Editing(Sender: TObject; Node: TTreeNode;
  var AllowEdit: Boolean);
begin
 oldv:=treeview1.Selected.Text;
 AllowEdit:=true
end;
```

Этот код запоминает еще "неиспорченное" пользователем значение переменной модели в глобальную переменную oldv типа STRING. А после занесения нового значения вызывается обработчик события OnEdited:

```
procedure TForm1.TreeView1Edited(Sender: TObject; Node: TTreeNode;
  var S: String);
begin
 try
  strtofloat(s)
 except
  MessageDlg(s+' - неверное значение размера',
  mtError,[mbOK],0);
  // восстанавливаем старое значение
  s:=oldv;
  abort
 end;
 SetLEngth(ch,Length(ch)+1);
 ch[length(ch)-1]:=node.AbsoluteIndex;
 // оживляем кнопку "Перестроить" - было изменение
 Button3.Enabled:=true
end;
```

Здесь в глобальный динамический массив ch (его элементы имеют тип WORD) заносятся абсолютные индексы (фактически порядковые номера) измененных переменных. Это понадобится, чтобы при перестроении модели обновлять значения только тех переменных, которые действительно изменились.

Кнопка "Перестроить" вызывает следующий обработчик:

```
procedure TForm1.Button3Click(Sender: TObject);
begin
 Modify;
```

```
 // сохраняем сборку
 doc.Save;
 button3.Enabled:=false;
 SetLength(ch,0)
end;
```

Основную работу выполняет процедура `Modify`:

```
procedure Modify;

var i:word;
    node,node1:TTreeNode;
    varname:STRING;

begin
 // цикл по массиву индексов измененных узлов в дереве
 for i:=0 to length(ch)-1 do
  with form1.TreeView1 do
   begin
    // ссылка на измененный узел (в нем новое значение переменной)
    node:=Form1.TreeView1.Items[ch[i]];
    // ссылка на родительский узел - в нем имя переменной
    node1:=node.Parent;
    varname:=node1.Text;
    // выкидываем комментарий в скобках
    if pos('(',varname)>0 then
     varname:=trim(copy(varname,1,pos('(',varname)-1));
     // в node1.parent.text - имя детали
     ChangeVar(node1.Parent.Text,
     node.Parent.Text,StrToFloat(node.Text))
   end
end;
```

6. Проверка корректности вводимых и расчетных значений

Не все комбинации значений переменных приводят к получению корректной модели. В рассматриваемом примере с гидроцилиндром очевидно, что величина `(dmax-dmin)/2` не может быть меньше минимально допустимой толщины стенки (в зависимости от рабочего давления это 2..10мм) и уж тем более недопустимо, чтобы `dmin1` оказался больше `dmin`.

Введение в программу кучи условий для проверки корректности значений переменных – занятие бесперспективное. Кроме того, при незначительном изменении требований к проектируемому изделию (скажем, гидроцилиндры стали делать из пластика и минимально допустимая толщина стенок теперь составляет 12.5мм) потребуется переделка алгоритма. Это явно тупиковый путь. Более разумно вынести все проверяемые условия во внешнюю базу данных, где они были бы записаны в понятном человеку виде и затем анализировались бы программой с применением аппарата формальных языков и грамматик. Тогда перед перестроением сборки расчетный модуль выполняет синтаксический и семантический разбор каждого условия и проверку его истинности. Если какое-то условие не выполняется, сборка не перестраивается и выдается соответствующее сообщение.

Прежде всего, создадим базу данных для хранения проверяемых выражений. Условимся, что все выражения должны быть булевского типа и при корректных значениях переменных должны вычисляться в True. База данных будет состоять из одной таблицы с простейшей структурой:

Имя поля	Тип поля	Длина поля
ID	автоинкрементное	-
Expression	Alpha	250

В поле Expression первой строки таблицы занесем выражение ((dmax-dmin)/2)>5. В расчетном модуле добавим компонент Table1 и настроим его на подключение к нашей таблице в момент запуска программы (событие OnCreate главной формы). Разумеется, на событии OnClose таблицу нужно закрывать.

Для выполнения лексического и синтаксического разбора можно использовать код калькулятора с переменными, который был рассмотрен в курсе "Лингвистическое и программное обеспечение САПР" и доступен для загрузки на сайте кафедры. К проекту нужно подключить файлы лексического анализатора parser.pas и синтаксического cals.pas. Лексический анализатор остается без изменений, а в синтаксическом надо поменять функцию GetVarValue, осуществляющую поиск значения переменной по ее имени. В исходной версии такой поиск выполняется в компоненте TStringGrid, а в нашей программе переменные и их значения хранятся в компоненте TTreeView. Поэтому в объекте TCalc заменим TStringGrid на TTreeView:

```
type
    TCalc=class
      private
       lexlist:TLexemList;
       lexnum:word;
       err:string;
       vrs:TTreeView;
      …
       public
      …
       constructor Create(sv:TTreeView);
```

При создании объекта TCalc ему на вход нужно подавать ссылку на компонент TreeView. Изменится и процедура поиска значения переменной. Мы знаем, что в дереве все имена переменных хранятся в узлах второго уровня, а значения – в узлах третьего. Поэтому следует организовать цикл по узлам второго уровня, и, при совпадении имени переменной с текущей лексемой, спуститься на уровень ниже (метод GetFirstChild узла дерева), чтобы взять из него значение переменной. Надо также помнить, что после имени переменной в скобках может храниться комментарий, который надо удалить. В итоге имеем следующий код:

```
function TCalc.GetVarValue:Extended;

var i:WORD;
    n:STRING;

begin
 FOR i:=0 TO Vrs.Items.Count-1 DO
  IF Vrs.Items[i].Level=2 THEN // это второй уровень?
   BEGIN
    n:=ansiuppercase(Vrs.Items[i].Text);
    // удаление комментария, если он есть
    if pos('(',n)>0 then
     n:=trim(copy(n,1,pos('(',n)-1));
    IF n=ansiuppercase(trim(GetLex.Lexeme)) then
```

```
      begin
       // на уровень ниже берем значение переменной
       Result:=StrToFloat(Vrs.Items[i].GetFirstChild.Text);
       exit
      end
     END;
   err:='Необъявленная переменная '+GetLex.Lexeme+' в позиции
'+IntToStr(GetLex.Pos)
  end;
```

В основной же программе создаем глобальные переменные для лексического и синтаксического анализаторов и на событии `OnCreate` создаем их (и тем более не забываем удалить эти объекты на событии `OnClose`):

```
USES … calc, parser,…
…

VAR
var
  Form1: TForm1;
  Lx:TLexicalAnalyzer;
  Cl:TCalc;
…


procedure TForm1.FormCreate(Sender: TObject);
begin
 …
 Lx:=TLexicalAnalyzer.Create;
 Cl:=Tcalc.Create(TreeView1);
 WITH Table1 DO
  BEGIN
    // путь к таблице
    DataBaseName:=ExtractFilePath(Application.ExeName);
    Open
  END
end;
```

Перед перестроением сборки проверяем корректность значений переменных при помощи функции `CheckVars`:

```
procedure TForm1.Button3Click(Sender: TObject);
begin
 if not(CheckVars) then
  abort;
 Modify;
 doc.Save;
 button3.Enabled:=false;
 SetLength(ch,0)
end;
```

Функция проверки выглядит следующим образом:

```
function CheckVars:boolean;

VAR r:STRING;

begin
 Result:=true;
 WITH Form1.Table1 DO // цикл по таблице
  BEGIN
   First;
   WHILE NOT(EOF) DO
    BEGIN
     // Лексический разбор
     Lx.Parse(Form1.Table1Expression.AsString);
     IF Lx.error<>'' THEN
      BEGIN
      MessageDlg('Ошибка в строке '+IntToStr(RecNo)+'
     таблицы:'+#13+Lx.error,mtError,[mbOK],0);
       Result:=FALSE;
       Exit
      END;
      // Семантический разбор
      r:=Cl.Evaluate(Lx.LexemList);
      IF Cl.error<>'' THEN
      BEGIN
      MessageDlg('Ошибка в строке '+IntToStr(RecNo)+' таблицы:'+#13+
       Cl.error,mtError,[mbOK],0);
       Result:=FALSE;
       Exit
      END;
     IF ANSIUpperCase(r)<>'TRUE' THEN
      BEGIN
       MessageDlg('Не выполнено условие'+#13+
       Form1.Table1Expression.AsString,mtError,[mbOK],0);
       Result:=FALSE;
       Exit
      END;
     Next
    END
   END
end;
```

Теперь расчетный модуль надежно защищен от недопустимых значений переменных.