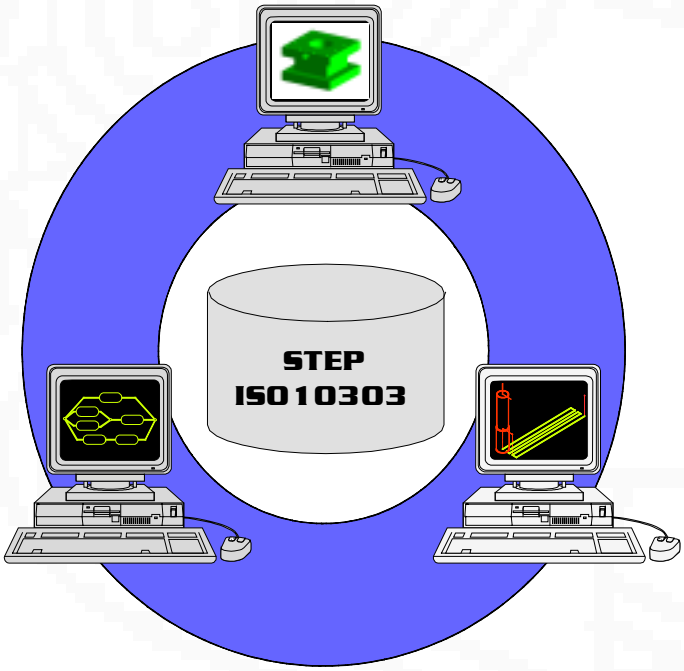
FESTEVAL

Development of a design environment with STEP processor for a feature based 3D-CAD System



Sebastian Leibrecht

Diploma thesis

written at the Universidade Methodista de Piracicaba, Brazil,
15.05.2000 – 11.08.2000

by *Sebastian Leibrecht*

Acknowledgement : *Prof. Dr.-Ing. Herbert Schulz, PTW, TU Darmstadt*

Supervising : *Prof. Dr.-Ing. Klaus Schützer, SCPM, Unimep*
Dipl. Ing. Caspar von Gyldenfeldt, PTW, TU Darmstadt
Dipl. Ing. Erik Claassen, DiK, TU Darmstadt

Project handling at SCPM : *Eng. Nara Gardini*

Enlisted institutes :

Laboratório de Sistemas Computacionais para Projeto e Manufatura
Universidade Methodista de Piracicaba
Santa Barbara d'Oeste, São Paulo, Brazil
Head of institute : Prof. Dr.-Ing. Klaus Schützer

Produktionstechnik und Spanende Werkzeugmaschinen
University of Technology Darmstadt, mechanical engineering
Darmstadt, Germany
Head of institute : Prof. Dr.-Ing. Herbert Schulz

Datenverarbeitung in der Konstruktion
University of Technology Darmstadt, mechanical engineering
Darmstadt, Germany
Head of institute : Prof. Dr.-Ing. Reiner Anderl

Prof. Dr.-Ing. Herbert Schulz
Produktionstechnik und
Spanende Werkzeugmaschinen

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Diplomarbeit

für

Herrn cand.- Ing. Sebastian Leibrecht

Konzeption und Implementierung eines Prozessors zwischen featurebasierten 3D-CAD-System und STEP-Format

Im Rahmen des EU-Forschungsprojekt FESTEVAL wird die Prozeßkette Konstruktion, Arbeitsplanung, Fertigung auf der Basis von Fertigungsfeatures verbessert. Ein wesentlicher Bestandteil des Projekts ist die Modellbeschreibung im neutralen Austauschformat nach STEP (ISO 10303), so daß Informationen zu den Features jederzeit entlang der Prozeßkette von den verschiedenen Programmmodulen abgerufen und modifiziert werden können.

Das Thema dieser Arbeit entstammt aus diesem Projekt. Es soll ein bestehendes Programm zur featurebasierten Konstruktion derart erweitert werden, daß es möglich ist, die in ihm erzeugten Daten im STEP-Format ablegen und einlesen zu können. Nach einer Prüfung des vorhandenen Programmbestands mit eventuellen Anpassungen an aktuelle Anforderungen soll ein Prozessor für das STEP-Format konzipiert und implementiert werden. Ein wichtiger Bestandteil der Arbeit ist die abschließende Überprüfung der Applikation durch die Modellierung eines gegebenen Werkstücks mit anschließendem Erzeugen einer Austauschdatei. Dadurch wird nachgewiesen, daß sich das erstellte Programm in die Prozeßkette von FESTEVAL eingliedern kann. Für alle Tätigkeiten soll begleitend eine angemessene Dokumentation erstellt werden.

Betreuer: Prof. Dr.-Ing. Klaus Schützer (PTW), Dipl.-Ing. Caspar v. Gyldenfeldt (PTW),
Dipl.-Ing. Erik Claassen (DiK)
Beginn der Arbeit: 15.05.2000
Abgabe der Arbeit: 18.08.2000

(Prof. Dr.-Ing. H. Schulz)

Summary

Topic of the EU research project *FESTEVAL* is the digital integration of all applications concerned with the process chain construction - planning - manufacturing.

A central aspect is the creation of a common database, in which all data created during the product life is stored. This database is used by all CAx systems in the process chain. A STEP datamodel is used as a basis for this database. The geometrical product definition is based on manufacturing features. Besides the geometric definition, the model also provides room for *FESTEVAL* specifific product information, like interdependencies between features and technological attributes.

This thesis is concerned with the development of a new design environment for the *FESTEVAL* process chain. This is done by extending the 3D CAD system Unigraphics with the required functionality. Besides the functionality for creating features, also a validation, the determination of technological and geometrical interdependencies and the ability to create STEP files is part of the new environment.

This thesis consists of the documentation of the development process for the new environment. In the beginning, a brief introduction into the basics of STEP and software development is given.

The most vital information of each chapter is framed.

Contents

1. Introduction

A large number of CAx systems were developed and implemented in the recent years to support all stages of product life by computer systems.
Most of these systems are specialized to support a certain application, and are based on an information model that handels the application specific view of the product. These CAx systems are not sharing a common database for the product information.

A large potential for development and research lies in the creation of a uniform and standardised format, for handling all information that is created during the different stages of product life.

This thesis and the underlying project is situated in this field.

1.1 *FESTEVAL*

Feature-based support for the development process chain
`design-planing-manufacturing´ *INCO-DC Project 96.2161*

The main focus of the research project *FESTEVAL* is the creation of a feature-based environment which supports all stages of the process chain. The basis for all parts of the project is the definition and implementation of a datamodel which handels all product data that is created by the different application modules.

The implemented datamodel is the central data structure that is used by all applications within the process chain. Aim of the project is the extension of existing CAx systems with the functionality needed to use the described datamodel. Thereby the systems are enabled to be digitally integrated into the process chain.
The datamodel should contain parts for the definition of features, technological information, machine concerned interdependencies and information for process planning.
The conecpt uses ISO 10303-224 as a basis for the specified datamodel. Modifications and extensions are done where required. The model is described in the data modeling language ISO 10303-11 (Express). The storage of data is done by physical files defined in ISO 10303-21.

The project is divided into four workpackages, each of which handels a certain topic. For each workpackage one of the participating institutes is responsible and is working on its topic. An overview over the project is given in fig. 1-1. It shows the workpackages, the topics of the

workpackages and the responsible institutes. The sketch in the center shows how the different stages of the process chain

- *design*
- *planning*
- *manufacturing*

are arranged around a common datamodel based on the STEP standard. The access to this common database from all systems concerned with the process chain is materialising the central idea of this project and of STEP.

The international research project *FESTEVAL* is successor of the research project ESPRIT III - #6090 - FIRES. It started at October 17th 1997 and ends on November 16th 2000.

1.2 Workpackage Unimep

Workpackage 1 is treated at the brazilian university Unimep in Santa Barbara d'Oeste/Saõ Paulo. Topic of this workpackage is the adaption of the 3D CAD system Unigraphics to the project requirements. This aim is reached by the creation of a new feature-based design environment within Unigraphics, which fulfills the demands of the project.

The design environment developed in workpackage 1 represents the beginning of the process chain, where geometrical and technological data is created. Within the design environment, all the required data has to be determined. To fulfil the idea of a common database, it must also be capable of storing the data in a database which is conform to the STEP datamodel used in *FESTEVAL*. Furthermore, it must be possible to create a physical file from this database for the exchange of the data to the other stages in the process chain.

The thesis at hand was done at Unimep and is therefore concerned with workpackage 1 of *FESTEVAL*. The main content is the development of the described design environment, thus a software development.

Parts of this thesis are :

- analysis of requirements
- planning of use-cases and user interface
- development of the structure of the software
- implementation of the control structures
- implementation of a STEP processor
- integration of existing structures into the software

FESTEVAL

EU - Research Project

Lab. Sistemas Computacionais para Projeto e Manufatura
Prof. Dr.-Ing. K. Schützer
FEMP - UNIMEP

Feature-based Support for the Development Process Chain 'design - planning - manufacturing'

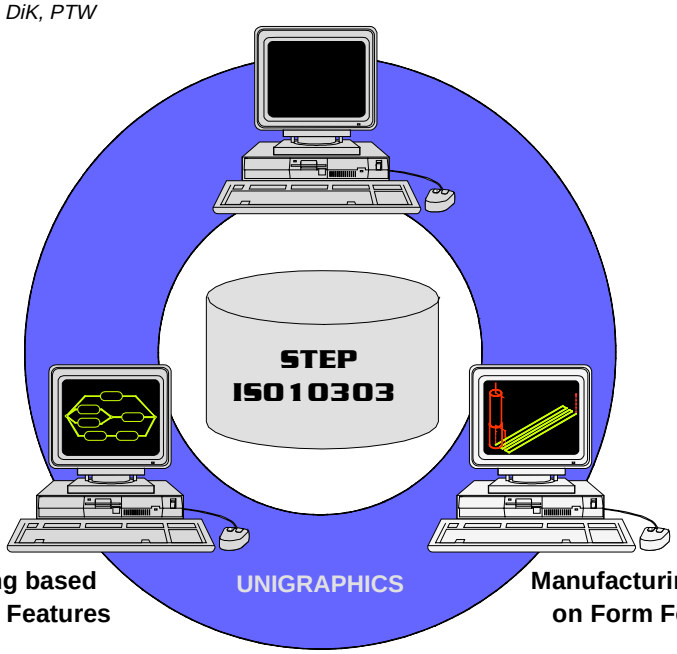
Workpackage 1

Módulo de Projeto Integrado baseado em Form Features. Reconhecimento de Interdependências Geométricas e Tecnológicas.
Responsável *- UNIMEP*
Parceiros *- Kade-Tech, DiK, PTW*

Workpackage 3

Especificação e Implementação do Modelo de Dados do Produto.
Responsável *- DiK*
Parceiros *- PTW, UNIMEP*

Design based on Form Features



Planning based on Form Features

Manufacturing based on Form Features

Workpackage 2

Módulo de Planejamento de Processo e Manufatura. Seleção e Sequenciamento de Operações de Manufatura.
Responsável *- PTW*
Parceiros *- Kade-Tech, DiK*

Workpackage 4

Especificação e Definição dos Requisitos Necessários. Testes do Protótipo.
Responsável *- Romi*
Parceiros *- PTW, SCPM*

Parceiros:

PTW DiK KADE-TECH SCPM ROMI ProSTEP

Colaborador:

Unigraphics Solutions

Contato: Engª. Nara Gardini

fig. 1-1 : overview FESTEVAL *(source : SCPM)*

The existing structures mentioned in the last point are methods which are developed by the SCPM (Unimep). They are handling topics in which expert knowledge about the process chain is implemented. This includes :

- creation of features in Unigraphics
- validation of features
- determination of interdependencies
- determination of additional information for the process chain

A rough listing of the tasks for workpackage 1 includes:

- creation of a FESTEVAL specific, feature-based design environment
- determination of all information required for the FESTEVAL datamodel
- validation of features (e.g. can it be manufactured ?)
- only successfully validated features can be used in the new environment
- determination of interdependencies
- creating of STEP exchange files with the product information

These steps are assuring that the data created in the design is usable for the further parts of the process chain. Since the geometric definition in the datamodel is based on manufacturing features, prismatic parts to be manufactured by tool machines can be handeled. An example part was designed to test the new environment. This part is shown in fig. 1-2.

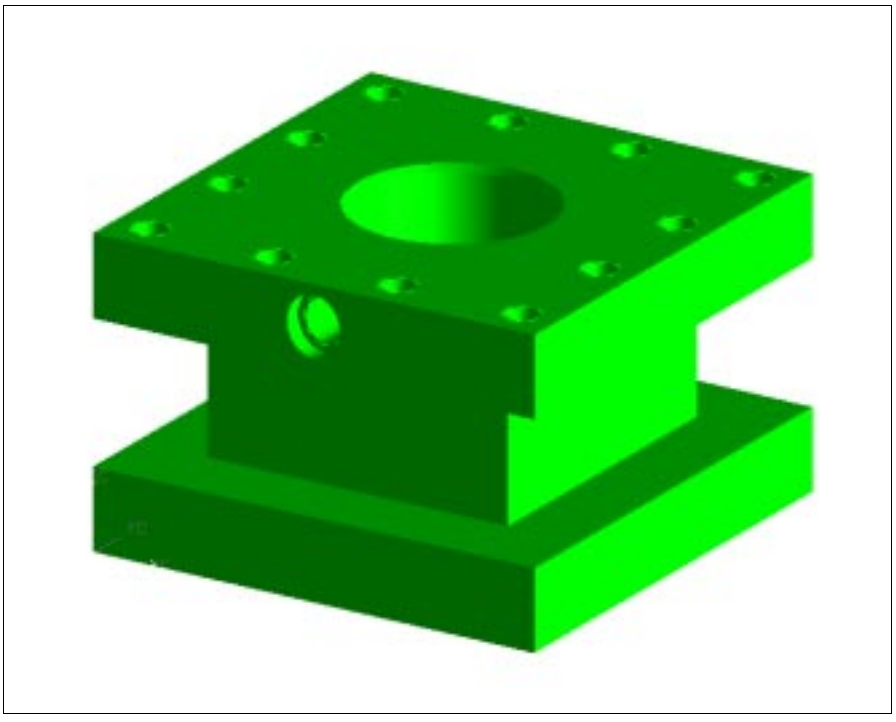


fig. 1-2 : example part

An overview of the functionality the new design environment should provide can be seen in fig. 1-3. The user is in contact with the environment via a new feature based user interface, which is based on the Unigraphics user interface. A number of feature based modeller functions can be choosen. Besides the inputs from the user, a number of further influences are reacting on these functions, like manufacturing know-how implemented in the software. The produced data is stored in a database which is conform to a feature based STEP datamodel.

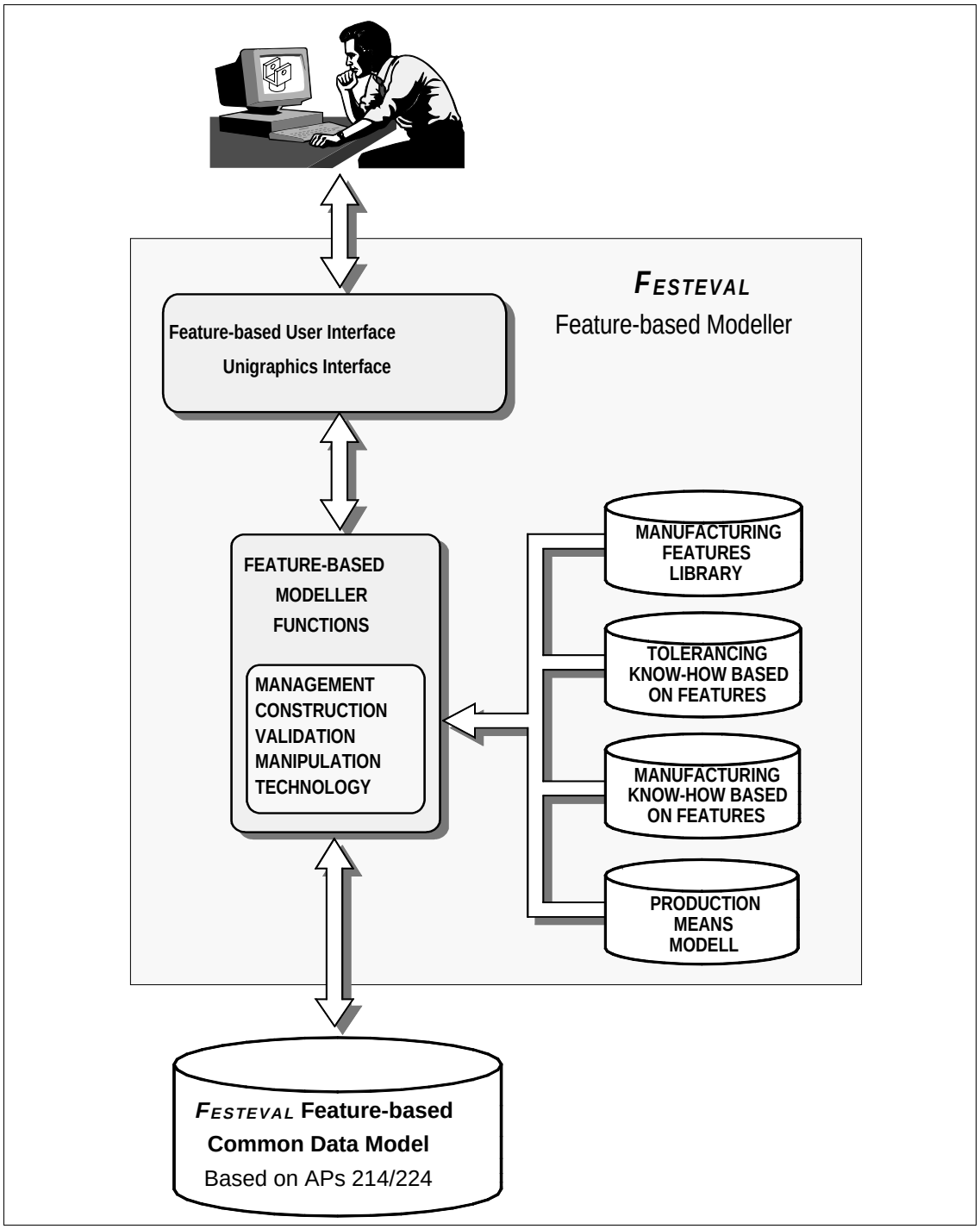


fig. 1-3 : FESTEVAL design environment (source : SCPM)

2. Basics

In this chapter the basic knowledge needed to understand this thesis is briefly explained. This mainly includes knowledge about STEP and about object-oriented software development.

Furthermore, an introduction about the used methodologies, notations and software tools is given.

2.1 Integrated product model / STEP

The term *integrated product model* denominates the handling of product information from all stages of product life and the different physical product properties in a uniform model. Considered are also the views from different applications.

Within ISO TC184/SC4, the integrated product model is developed under the description ISO 10303 Product Data Representation and Exchange. It is well known by the name STEP (**St**andard for the **E**xchange of **P**roduct Model Data).

Besides the specification of the product model with base and application data, also the methods for description, implementation and conformance testing are defined in STEP. Also included in the norm are application protocols for the presentation of application specific views of the product model.
The STEP standard is described in a large number of ISO documents. Each document describes a special, self-contained part of the standard. Examples are the description of a *notation language*, a *method* or of a specific *application protocol*.
The documents are arranged in series. Each series represents a functional section of the standard. Examples for series are the *description methods* (notations), *implementation methods* or *application protocols*.
Each document can be identified by a number. The first number together with the amount of digits is specifying the series to which a document belongs. Example : document 11 belongs to the 1x series (*description methods*), document 224 belongs to the 2xx series (*application protocols*).

In fig. 2-1 an overview of the series and their documents can be seen.

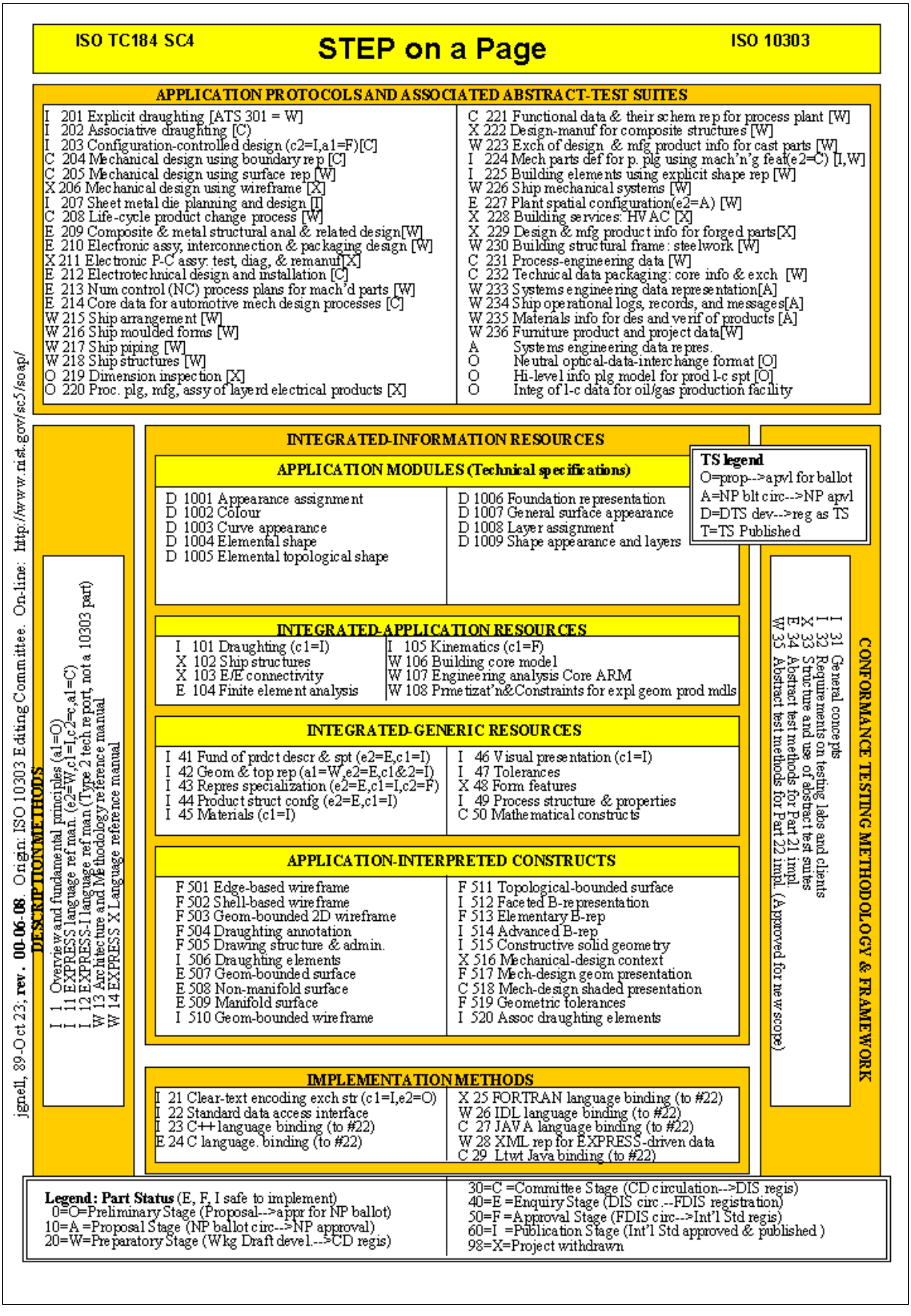


fig. 2-1 : ISO 10303 (STEP) documents *(source : NIST)*

Within this thesis, the following documents are of importance:

ISO 10303 –

11 Express / Express-G
21 file format for STEP files
22 data access interface (SDAI)
23 language binding of SDAI in C++
224 application protocol

The topics and contents of these documents will be explained in the following sections.

further readings : [And2000], [ISO10303]

2.1.1 Express-G / Express

> ISO 10303-11 defines the formal description language Express. With Express, the product model can be described unequivocally.

The structure of Express partly originates from object-oriented models and partly from entity-relationship models. A structure that is defined in Express can be created and edited with ASCII editors. Through the uniform structure of the files they can be used to transfer model specific information between different systems.
Express-G defines a graphical representation of Express. This makes the use of graphical modelling for Express structures possible. There are several software tools for the creation of Express-G diagrams. The diagrams can be saved in Express files to transfer the model information to other systems.
A structure described in Express is called schema.

In fig. 2-2a the main constructs of Express-G are shown. Their utilisation is as follows (examples are shown in fig. 2-2b) :

entity

An entity represents the abstract definition of a piece of data that is considered in the model. An entity is used to instanciate (create) a concrete entry in a database.

Example : An entity `block` *can be used to create an entry for a block in the database.*

abstract entity

An abstract entity can not directly be used to create entries in the database. It only serves as a supertype for concrete or abstract subtypes.

Example : An abstract enity `(ABS)shape` *is not needed to create entries in the database, because each baseshape in the database has to be of a concrete subtype, like* `block` *or* `cylinder`*.*

attribute / attribute relation

An attribute is representing the actual information an entry in the database can hold. It is of simple types like *real*, *integer* or *string*.

Example : An entity `block` *has three real-attributes :* `length`*,* `width` *and* `height`*.*

enity relation

Relations between entities are representing the connections between the entries in the datamodel. If an entity has a relation to another, it means either that one is a part of the other (unidirectional) or that they belong together (bidirectional).

Example : An entity `figure` *consists of a number of entities* `shape`*.*

supertype – subtype

The subtype-entity of such a relation inherits (has without explicit definition) all the relations of its supertype-entity.

Example : Each subtype-entity of `shape` *(*`block`*,* `cylinder`*) has an attribute* `shape_id`*.*

See fig. 2-2b for the Express-G diagram representing the examples used here.

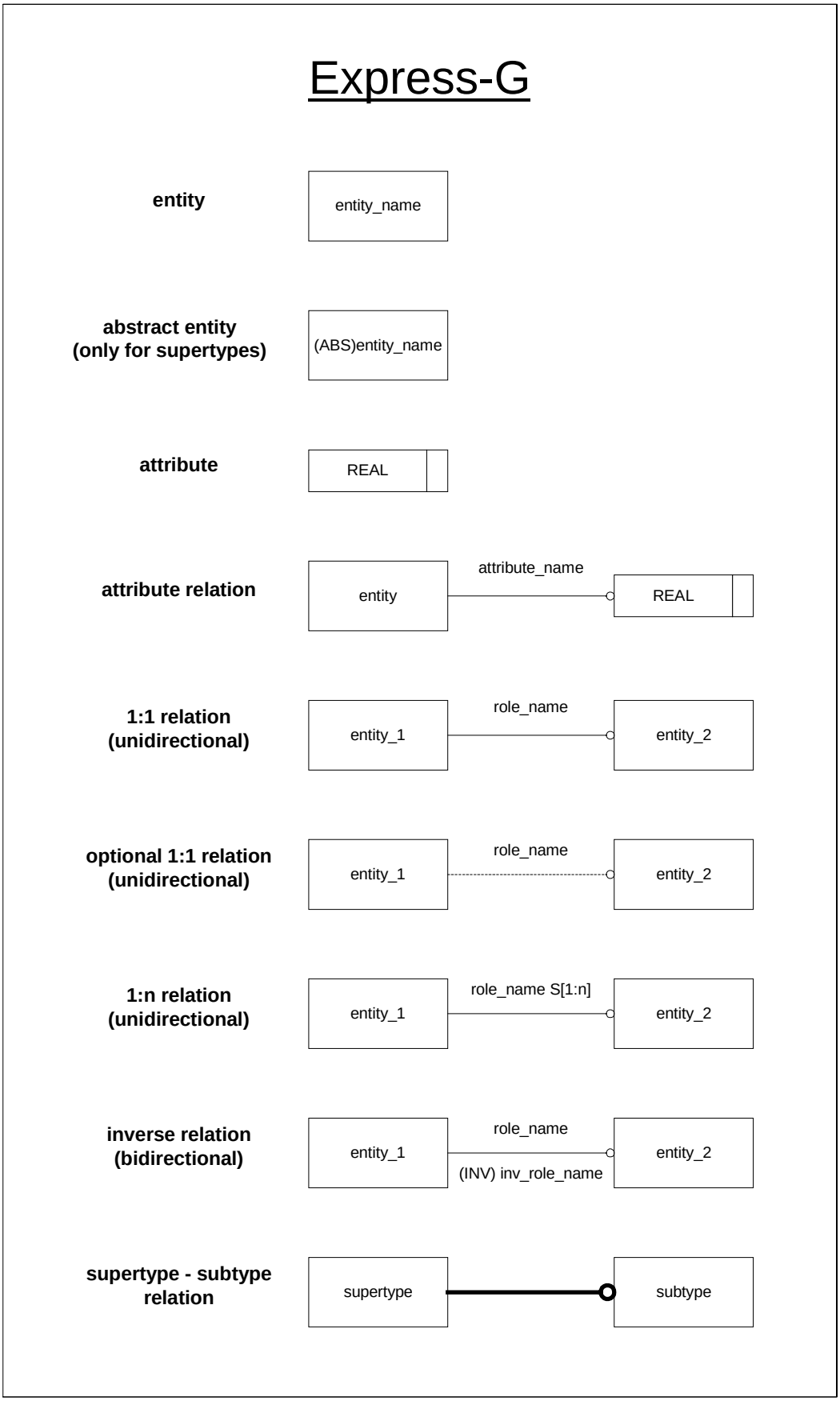


fig. 2-2a : main constructs of Express-G

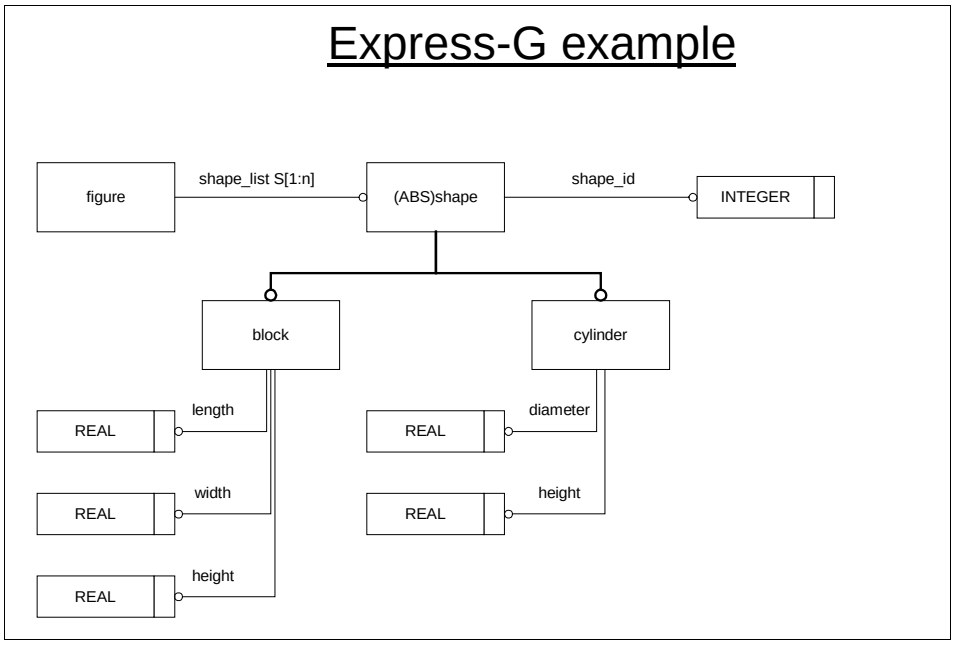


fig. 2-2b : Express-G example

2.1.2 STEP Exchange File

A structure described in Express represents an abstract model. It defines the semantics and structure of data. The purpose of the model is the instanciation of entities. Thereby the model is ‚filled' with data and information about a certain product is pictured in the model.
To transfer the product information to another system, or to store the information, a physical ASCII file can be created.

ISO 10303-21, STEP Exchange File Format, describes the structure of such a file.

The fixed structure allows different Systems to read the file and correctly interpret the stored information. This is the key to one of the main aims of STEP, the exchange of product information between different systems.
An example for a STEP file can be seen in fig. 2-3. Each entry has an index. The piece of information following the index is the type of the entry, thus the entity used to create the entry. The following list holds the relations of this entry. A relation is either a reference to another entry (entity relation) or an attribute, which is represented by its value. Entries are always referenced by other entries via the index.

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION(('STEP Physical File'),'Level 1.0');
FILE_NAME('D:\FESTEVAL\UGParts\block_hole.stp','2000-07-
10T11:09:06',('Current User'),('DiK'),'','','');
FILE_SCHEMA(('HSCWF'));
ENDSEC;
DATA;
#0=PART($,$,#2,$,$,$,$,$,$,#1,$);
#1=FEATURE_LIST((#88));
#2=SHAPE($,$,#3);
#3=BLOCK_BASE_SHAPE(#5,#4,#20,#21);
#4=NUMERIC_PARAMETER('Base_Shape_Length','mm',200.);
#5=ORIENTATION(#6,(#11,#16));
#6=CARTESIAN_POINT(0,3,(#7,#8,#9));
#7=NUMERIC_PARAMETER('x','mm',0.);
#8=NUMERIC_PARAMETER('y','mm',0.);
#9=NUMERIC_PARAMETER('z','mm',0.);
#10=DIRECTION($,$,(#12,#13,#14));
#11=VECTOR($,3,#10,$);
#12=NUMERIC_PARAMETER('x','mm',1.);
#13=NUMERIC_PARAMETER('y','mm',0.);
#14=NUMERIC_PARAMETER('z','mm',0.);
#15=DIRECTION($,$,(#17,#18,#19));
#16=VECTOR($,3,#15,$);
#17=NUMERIC_PARAMETER('x','mm',0.);
#18=NUMERIC_PARAMETER('y','mm',1.);
#19=NUMERIC_PARAMETER('z','mm',0.);
#20=NUMERIC_PARAMETER('Width','mm',100.);
#21=NUMERIC_PARAMETER('Height','mm',50.);
#22=ORIENTED_EDGE(468,$,#23,.T.,$);
#23=EDGE_CURVE($,.T.,#24);
#24=CIRCLE(467,1,#25,15.);
#25=AXIS2_PLACEMENT_3D($,$,#26,(#30,#34,#38),$,$);
#26=CARTESIAN_POINT(0,3,(#27,#28,#29));
#27=NUMERIC_PARAMETER('x','mm',-96.0772547548711);
#28=NUMERIC_PARAMETER('y','mm',43.6737938128487);
#29=NUMERIC_PARAMETER('z','mm',0.);
#30=DIRECTION($,$,(#31,#32,#33));
#31=NUMERIC_PARAMETER('Circle_Placement','mm',-1.);
#32=NUMERIC_PARAMETER('Circle_Placement','mm',0.);
#33=NUMERIC_PARAMETER('Circle_Placement','mm',0.);
#34=DIRECTION($,$,(#35,#36,#37));
#35=NUMERIC_PARAMETER('Circle_Placement','mm',0.);
#36=NUMERIC_PARAMETER('Circle_Placement','mm',1.);
#37=NUMERIC_PARAMETER('Circle_Placement','mm',0.);
#38=DIRECTION($,$,(#39,#40,#41));
#39=NUMERIC_PARAMETER('Circle_Placement','mm',0.);
#40=NUMERIC_PARAMETER('Circle_Placement','mm',0.);
#41=NUMERIC_PARAMETER('Circle_Placement','mm',-1.);
#42=ORIENTED_EDGE(469,$,#43,.T.,$);
#43=EDGE_CURVE($,.T.,#44);
#44=CIRCLE(480,1,#45,15.);
#45=AXIS2_PLACEMENT_3D($,$,#46,(#50,#54,#58),$,$);
#46=CARTESIAN_POINT(0,3,(#47,#48,#49));
#47=NUMERIC_PARAMETER('x','mm',-96.0772547548711);
#48=NUMERIC_PARAMETER('y','mm',43.6737938128487);
#49=NUMERIC_PARAMETER('z','mm',-50.);
#50=DIRECTION($,$,(#51,#52,#53));
#51=NUMERIC_PARAMETER('Circle_Placement','mm',-1.);
```

fig. 2-3a : example STEP file (part 1)

```
#52=NUMERIC_PARAMETER('Circle_Placement','mm',0.);
#53=NUMERIC_PARAMETER('Circle_Placement','mm',0.);
#54=DIRECTION($,$,(#55,#56,#57));
#55=NUMERIC_PARAMETER('Circle_Placement','mm',0.);
#56=NUMERIC_PARAMETER('Circle_Placement','mm',1.);
#57=NUMERIC_PARAMETER('Circle_Placement','mm',0.);
#58=DIRECTION($,$,(#59,#60,#61));
#59=NUMERIC_PARAMETER('Circle_Placement','mm',0.);
#60=NUMERIC_PARAMETER('Circle_Placement','mm',0.);
#61=NUMERIC_PARAMETER('Circle_Placement','mm',-1.);
#62=ORIENTED_FACE($,$,$,#81);
#63=CYLINDRICAL_SURFACE($,$,#64,15.);
#64=AXIS2_PLACEMENT_3D($,$,#77,(#65,#69,#73),$,$);
#65=DIRECTION($,$,(#66,#67,#68));
#66=NUMERIC_PARAMETER('Axis','mm',0.);
#67=NUMERIC_PARAMETER('Axis','mm',0.);
#68=NUMERIC_PARAMETER('Axis','mm',1.);
#69=DIRECTION($,$,(#70,#71,#72));
#70=NUMERIC_PARAMETER('y-Direction','mm',0.);
#71=NUMERIC_PARAMETER('y-Direction','mm',0.);
#72=NUMERIC_PARAMETER('y-Direction','mm',1.);
#73=DIRECTION($,$,(#74,#75,#76));
#74=NUMERIC_PARAMETER('Norm_dir','mm',0.);
#75=NUMERIC_PARAMETER('Norm_dir','mm',0.);
#76=NUMERIC_PARAMETER('Norm_dir','mm',-1.);
#77=CARTESIAN_POINT(0,3,(#78,#79,#80));
#78=NUMERIC_PARAMETER('x','mm',96.0772547548711);
#79=NUMERIC_PARAMETER('y','mm',43.6737938128487);
#80=NUMERIC_PARAMETER('z','mm',25.);
#81=FACE_SURFACE($,(#82,#83),.T.,#63,#84);
#82=EDGE_LOOP($,.T.,1,(#22));
#83=EDGE_LOOP($,.T.,1,(#42));
#84=CARTESIAN_POINT(0,$,(#85,#86,#87));
#85=NUMERIC_PARAMETER('x','mm',96.0772547548711);
#86=NUMERIC_PARAMETER('y','mm',43.6737938128487);
#87=NUMERIC_PARAMETER('z','mm',25.);
#88=ROUND_HOLE($,$,#92,#109,#108,#89,#91,$,#107);
#89=CIRCULAR_CLOSED_PROFIL($,#90);
#90=NUMERIC_PARAMETER('Diameter','mm',30.);
#91=LINEAR_PATH(#92,$);
#92=ORIENTATION(#93,(#98,#103));
#93=CARTESIAN_POINT(0,3,(#94,#95,#96));
#94=NUMERIC_PARAMETER('x','mm',96.0772547548711);
#95=NUMERIC_PARAMETER('y','mm',43.6737938128487);
#96=NUMERIC_PARAMETER('z','mm',50.);
#97=DIRECTION($,$,(#99,#100,#101));
#98=VECTOR($,3,#97,$);
#99=NUMERIC_PARAMETER('x','mm',0.);
#100=NUMERIC_PARAMETER('y','mm',0.);
#101=NUMERIC_PARAMETER('z','mm',-1.);
#102=DIRECTION($,$,(#104,#105,#106));
#103=VECTOR($,3,#102,$);
#104=NUMERIC_PARAMETER('x','mm',0.);
#105=NUMERIC_PARAMETER('y','mm',0.);
#106=NUMERIC_PARAMETER('z','mm',0.);
#107=THROUGH_BOTTOM_CONDITION();
#108=NUMERIC_PARAMETER('EdgeRadius','mm',0.);
#109=DESCRIPTIVE_PARAMETER($,$);
ENDSEC;
END-ISO-10303-21;
```

fig. 2-3b : example STEP file (part 2)

2.1.3 Standard Data Access Interface (SDAI)

Product information can be stored in different systems. The way of accessing the data is defined in the Standard Data Access Interface (SDAI).

> The SDAI defines the functionality of a programming interface or a toolbox for the acces to product information to support the development of STEP based applications.
> The SDAI is defined in ISO 10303-22.

The SDAI is an abstract definition. Further definitions for the implementation in certain programming languages are provided in ISO 10303-23 and onwards. ISO 10303-23 defines the implementation of the SDAI in C++. Together with tith this the SDAI defines the interface between the Express model (a pure data model) and an object-oriented model for the implementation in C++.

2.1.4 Development of product and application models

STEP supplies a number of methods for the development of product and application models.

> Application protocols are specifying a subset of product data that is relevant for a certain application. The product model is specialized for a certain application.
> Application protocols are defined in the 200 series (ISO 10303-2xx).

The development of application protocols includes the development of the following part models in the given order (see also fig. 2-4) :

a) Application Activity Model (AAM)

The process chain is specified in this model. The activities producing or processing process data are the main content of this model. It is described in process modelling language IDEF0.

b) Application Reference Model (ARM)

This model defines the semantics and structure of the data. It is described in the data modelling language Express/Express-G.

c) Application Interpreted Modell (AIM)

This model describes how the constructs of the model are used. The model is created from the ARM by using certain imaging rules. It is described in Express/Express-G.

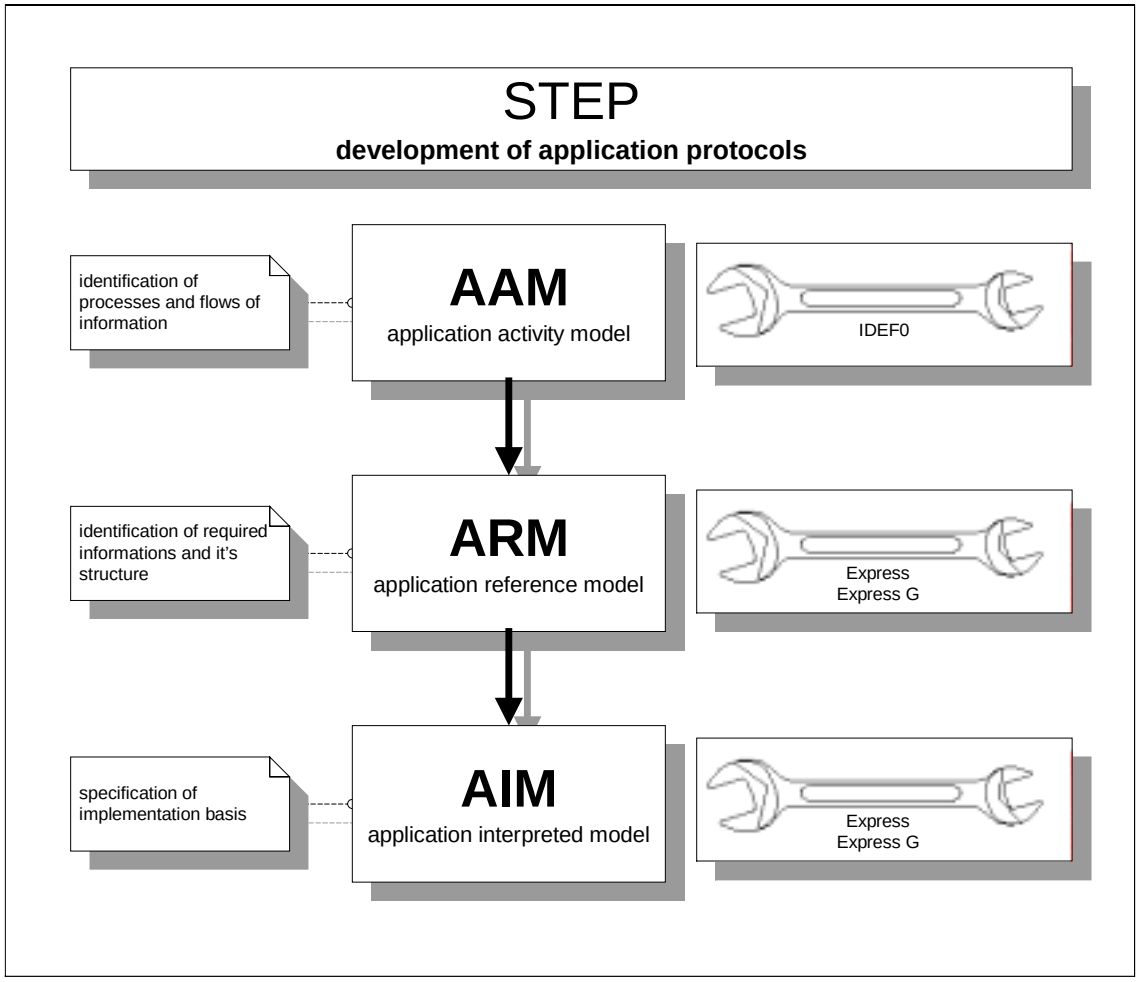


fig. 2-4 : development of STEP application protocols

2.1.5 AP224

AP224 is the application protocol defined in ISO10303-224 :
"Mechanical product definition for process plans using machining features"

The focus of AP224 is the geometric description of the product on the basis of manufacturing features. It is the foundation for the datamodel used in *FESTEVAL*. It is adapted to the special requirements of *FESTEVAL* where needed.

Not a completely new application protocol with AAM, ARM and AIM is developed for *FESTEVAL*, but the ARM of the existing AP224 is adapted.

2.2 Object-oriented software development

The content of this thesis is mainly a software development process. The basic knowledge about modern methodics and notations for software development are decribed in this section.

further readings :[Oest97], [Booch2000], [UML99], [Eder2000], [Fow98], [Lb2000]

2.2.1 Object-orientation

> Object-orientation is the approach to combine data and functionality in the same units. Objects existing in the real world are imaged in an abstract model.

The basic characteristics of objects are :

- objects can contain information (data) : attributes
- objects can provide a functionality : methods
- objects can be used only via certain specified interfaces (black box principle)
- objects can know other objects to communicate
- objects are units that are self responsible for their data and conformity

An object-oriented system consists of a number of objects with different kinds of relations. The structure of an object is defined in its class. Each object is an instance of a class.

The two main relations in object-oriented models are :

Aggregation (object relation) :

An object has another object as an attribute. It knows it and can use it via its interfaces.

Generalization (class relation) :

A class (sub-class) inherits attributes and methods from another class (super-class). The sub-class can define additional attributes and methods or overwrite methods. An instance of the sub-class can be used whereever an instance of a super-class could be used.
Generalization is commonly used together with abstract classes (see further readings).

The superiority of object-oriented systems is evident. The self responsibility of the objects makes the debugging and maintenance easy. Objects can be redefined easily without influencing the rest of the system because of the black block principle. The system can easily be extended.
But a good model, that has all these advantages, has to be planned very carefully.

In the previous years a lot of powerful tools for the development of object-oriented models were created. This includes methodologies for the analysis of the real world as well as notations for the models and lots of other helpful tools.

The counterpart to object-orientation is procedure-orientation. In procedure-oriented programming the main focus is not the creation of a model of the real world but on the procedures needed to solve a problem.

Example : The volume of a box is to be calculated by a computer program.

Procedure-oriented solution : A function for the calculation of boxes is implemented. With the call of the function the dimensions of the box have to be provided to the function as parameters. The function calculates the volume using the given dimensions and returns the result to the caller.

Object-oriented solution : A class `box` *for the instanciation of objects is provided. These objects are representing boxes. Each object of this class has its own dimensions, which can be defined during the instanciation or with special methods. The object provides a method which calculates the volume using its own dimensions.*

In object-oriented systems each function and variable belongs to an object. Functions are called *methods*, variables *attributes*. Together they are called *classmembers*. To acces classmembers the name of the owning object must be specified.

Exception : An object accessing classmembers of itself must not necessarily specify itself as owning object (`this.`).

Examples : Access to classmembers

```
volume = a_box_object.get_volume();
(this.)heigth = 10.0;
```

Procedure-oriented software development is loosing more and more significance and gets displaced by object-oriented software development in many fields.

Allthough an object-oriented design of the software is approached in this thesis, there are some cases where also procedure-oriented design has to be used. An example is the programming interface of Unigraphics (UG/Open), which is used extensively in the project. It is completely procedure-oriented and has to be included of the object-oriented design.

2.2.2 Unified Modelling Language

Since the early 90‘s a lot of methods for the design of object-oriented models have been developed. The methods of Booch and Rumbaugh were the ones with the biggest propagation. The unition of these resulted in the OMT (Object Modelling Technique) which included a notation and a methodology. In 1995 Booch, Rumbaugh and Jacobsen startet to work on a unified standard. The result is the UML (Unified Modelling Language), which was standarized in 1997.

> The UML defines the notation and a meta model for object-oriented systems.
> A methodology is not included in the UML.

The methodology used in this thesis is based on the OMT and the potential of the UML. The most commonly used an probably most important diagram of the UML ist the class diagram.

UML diagrams used in this thesis are :

class diagrams : They are describing the static structure of object-oriented models. They are used to plan the classes, classmembers and relations of the system. Parts of class diagrams are classes, classmembers and relations. The most important constructs of class diagrams are shown in fig. 2-5. See also chapter 2.2.1 for further explanations.

use-case diagrams : They are used for the presentation of use-cases from the view of the users of the software. Expectations, requirements and the GUI can be determined by the analysis of use-cases. Constructs of use-case diagrams are actors, use-cases and relations.

activity diagrams : They are describing the activities which the software can or should perform. This diagram can be used for describing different kinds of circumstances on different levels of detail. Constructs of activity diagrams are activities, decisions, flows and responsibilities.

sequence diagrams : The dynamic behaviour of the object system during run-time can be described using sequence diagrams. Their focus are the interactions between objects. The sequence diagrams are defining the major part of the dynamic model. Constructs of sequence diagrams are objects, messages and life-lines.

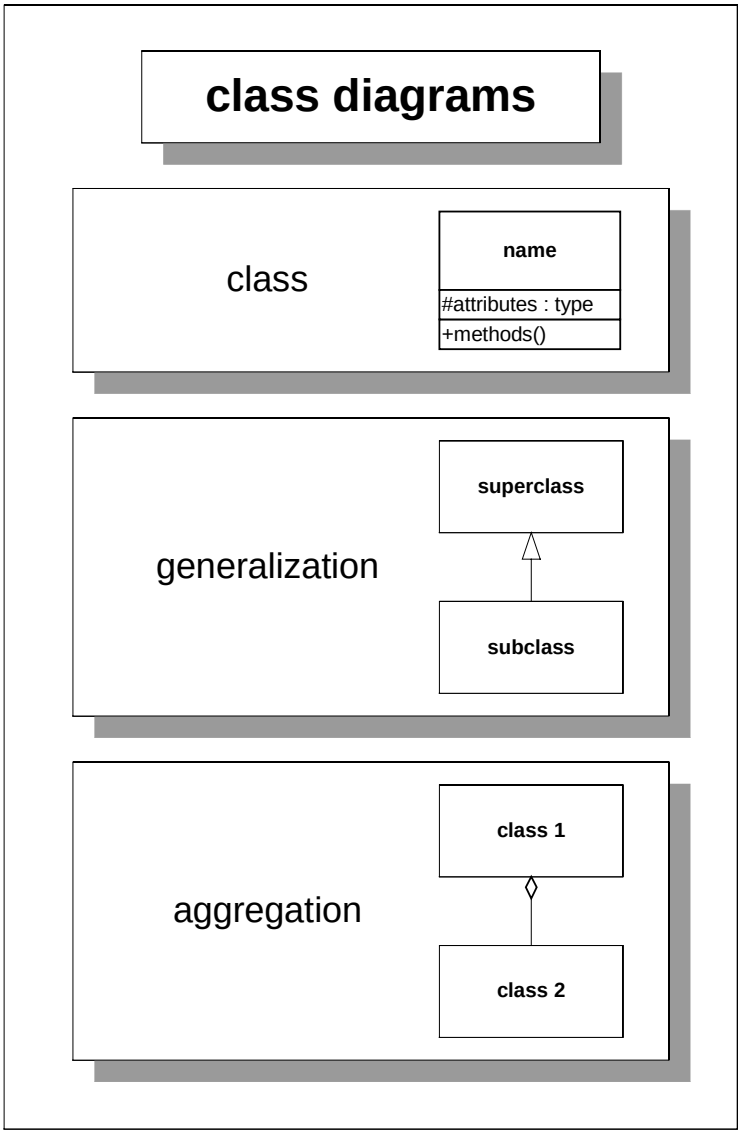


fig. 2-5 : important constructs of class diagrams

2.2.3 The software development cycle

The methodology used in this thesis divides the software development cycle into four stages :

- analysis
- design
- implementation
- use

These steps are following neither in themselves nor in the whole a certain sequence. *„The object-orientated system development follows an evolutionary, iterative approach model. ...*

The development from the rough to the detailed takes not place synchronously for all parts of the system but dependend on the situation ...“ [Oest97].

The stages of the software development are not creating a procedure with a beginning and an end, but a cycle. This means that the software development never necessarily ends, and that the software is never necessarily finished and perfect (like every product). A good technique is to go through the stages very quickly in the beginning to create prototypes with restricted functionality. In further traverses of the cycle, these prototypes can be extended until the needed functionality is included.
In the following sections the different stages of software development are briefly explained.

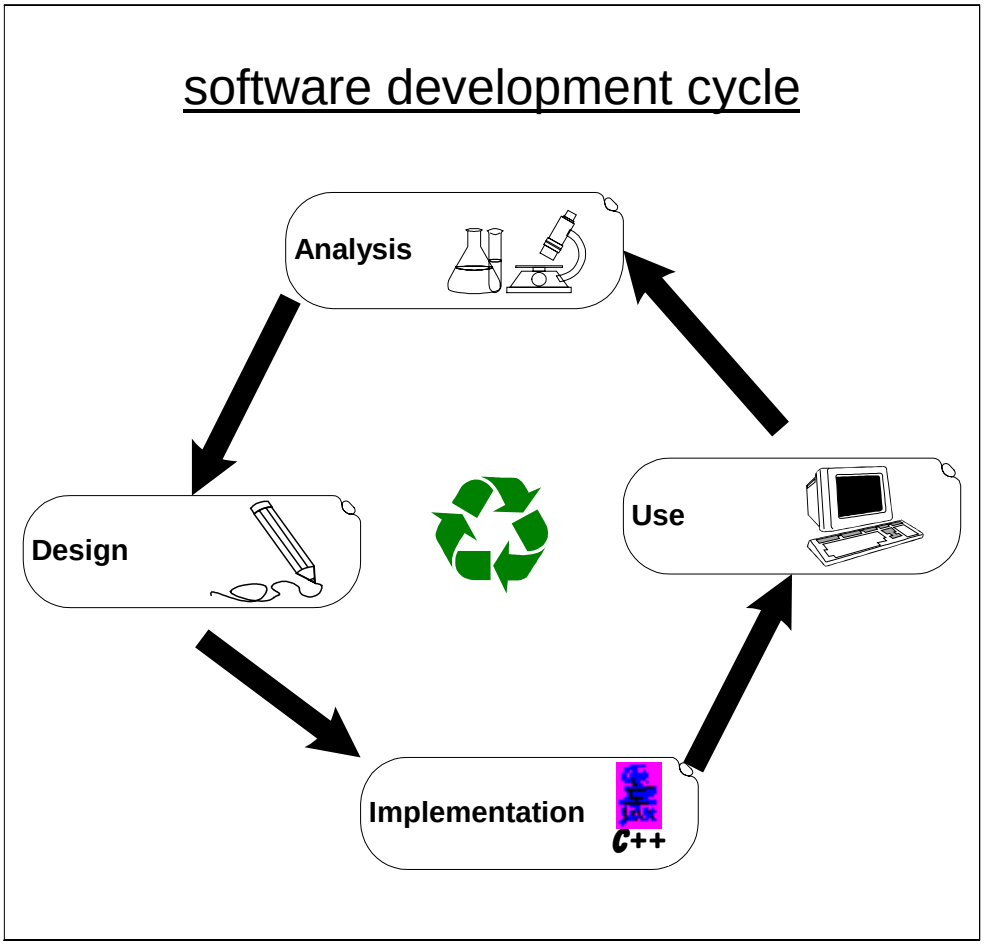


fig. 2-6 : software development cycle

2.2.3.1 Analysis stage

The focus of the analysis stage is the determination and investigation of the requirements and aspects of the real world.

Some analyses are :

- which processes should be supported by the software
- which (part) solutions are already existent
- what interfaces does the system need
- what are the enlisted people expecting from the software
- where are future extensions important

A major part of the analysis stage can be accomplished by the examination of use-cases and their resulting intenal processes.

diagrams : use-case diagrams, activity diagrams

2.2.3.2 Design stage

In the design stage the expressions and circumstances of the real world are transferred into an abstract model. This model has to fulfil a number of requirements.
Besides the requirements resulting from the analysis stage, there are also some basic requirements like clarity, maintainability, extendability and robustness. Also it must satisfy the aspects of object-orientation. The main parts of an object-oriented model are :

- basis elements (classes, objects, attributes, methods, ...)
- static elements (associations, aggregations, generalizations, ...)
- dynamic elements (conditions, activities, ...)

The design is the most productive stage of the software development in view of the final result (the software).

diagrams : class diagrams, interaction diagrams (sequence diagrams)

2.2.3.3 Implementation stage

A complete and carefully created model can be transferred directly into any object-oriented programming language. The classes are individually coded and gradually brought together to the complete system according to the model.

diagrams : structograms (Nassi-Schneidermann)

2.2.3.4 Use stage

The use stage is as much part of the development cycle as any other stage. The software is used in a testing environment with prototypes, or commercially by the final user with release versions.

The main results of this stage are information about :

- bugs (errors)
- new requirements for the functionality
- new requirements for the GUI

These results are direct inputs to the analysis stage of the next traverse of the cycle.

2.3 Software tools

In this project a lot of software tools are employed. The most important of these tools are introduced in this section. The functionality and handling is explained briefly where necessary.

2.3.1 Unigraphics

Unigraphics is a modern 3D-CAD system. It is available for different platforms (Windows NT, Unix). A three dimensional product shape can be created via a graphical user interface.

The created structure consists of parts, which can be united to assemblies. The shape of a part is mainly defined by features. Examples for features are holes, pockets and chamfers (removing material) and bosses and pads (creating material). Besides features, freeform shapes are important modelling structures. Unigraphics uses an own datamodel and file format for handling the product information. It is not STEP conform.
A screenshot of Unigraphics can be seen in fig. 2-7. It shows an opended part in the display window and a dialog box for creating threads.

further readings : [Ugmn98]

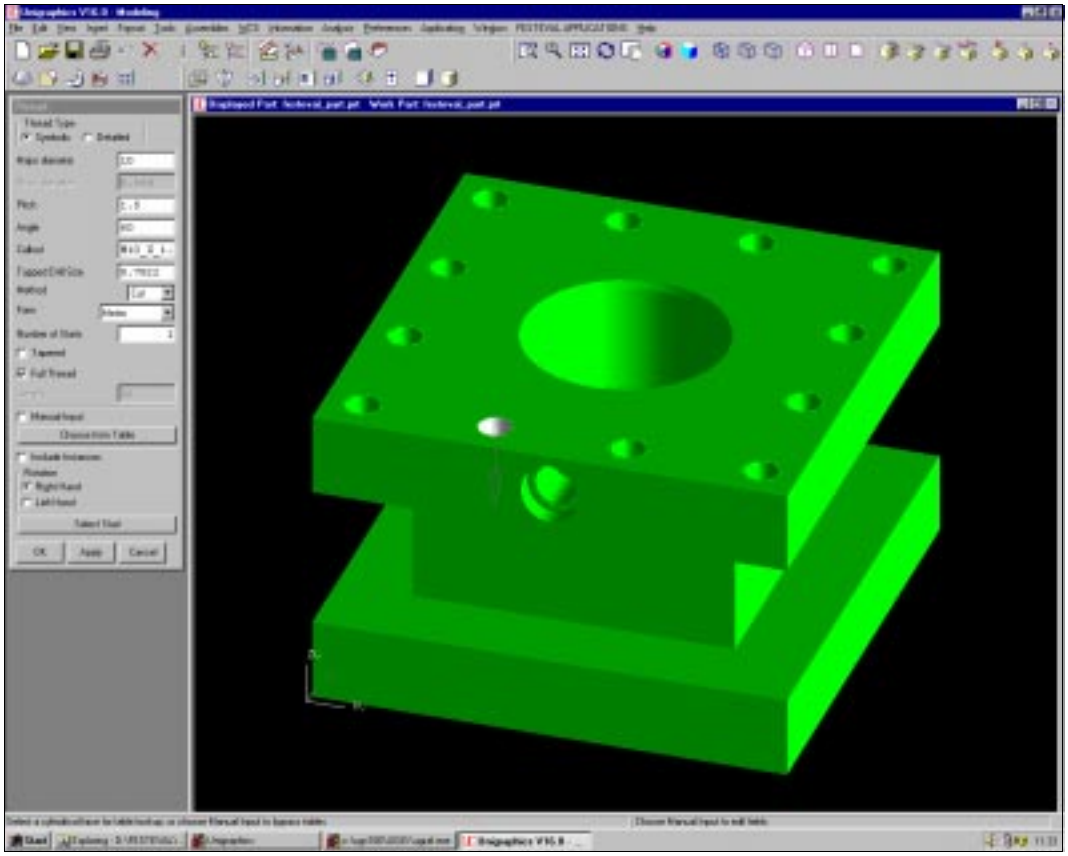


fig. 2-7 : screenshot Unigraphics

2.3.2 UG / Open

Some software packages are providing a programming interface for access to system internal functionality. These interfaces are called API (application programmers interface) and they allow the access and manipulation of the database and user interface. This allows the creation of new environments and functionality.
In Unigraphics this API is called UG/Open. It is based on the programming language C++.

Each instance of the database can be identified by an index pf the type `tag_t`. Such instances can be features for example. For the creation, reading and manipulation of instances the functions of the group `UF_MODL_*` are used. For the access to the user interface the functions of the group `UF_UI_*` are used.
For a detailed explanation of the functions the documentation of UG/Open should be referred.

further readings : [Ugapi98]

2.3.3 NIST STEP Class Library (SCL)

In 1988 the department National PDES Testbed was founded at the National Institute of Standards and Technology (NIST) in the USA. The expression PDES (Product Data Exchange using STEP) describes the activities of the USA for the development of STEP. The National PDES Testbed provides a neutral testing environment for the STEP development, in which the quality of the datamodels and the application of new STEP technology is examined.

> The NIST STEP Class Library is a collection of libraries for the development of STEP software. It was created at the National PDES Testbed and has the aim to support and establish the industrial use of STEP. It is available freely and is implemented in the programming language C++.

The basis of the SCL is the SDAI (ISO 10303-22) and its implementation rules for C++ (ISO 10303-23). The SCL also provides functionality for creating STEP exchange files (ISO 10303-21). The handled data model must exist as an Express schema (ISO 10303-11). This can for example be the reference model (ARM) of an application protocol (ISO 10303-2xx).
Within FESTEVAL the functionality of the SCL is not accessed directly, but via another library. For possible extensions of this library an understanding of the SCL is necessary.

further readings : [Lay90], [Mor92], [Saud95], [Schzk99]

2.3.3.1 Brief guide to SCL

The SCL consists of the following libraries :

- STEP Schema Class Library
- STEP Core Class Library
- Registry Classes
- Data Editor Library

2.3.3.1.1 STEP Schema Class Library

This is the model and schema specific part of the SCL. It has to be created for each schema. It represents the content of the Express schema as C++ classes. The Schema Class Library can be created automatically from Express Code by using the SCL tools Fedex Plus or software that uses Fedex Plus (e.g. WinStep).

For each Express Schema the following files are created :

- header file : class definitions
- library file : class implementations
- initialization file : function for initializing

The filenames consist of the name of the scha with a preceding SDAI and an identifier. Each attribut in the Express Schema is represented by a protected attribute in the C++ (SDAI) class. The acces to the attributes is done by public methods (information hiding). The Express schema is linked to the library at compilation time (early binding).

2.3.3.1.2 STEP Core Class Library

The STEP Core Class Library is the the schema independent basis for the Schema Class Library.

The most important classes are :

`STEPentity`

Abstract basis class for all classes that are representing entities of a schema. It is the root in the inheritance hierarchie of the entities.

The classmembers of `STEPentity` are :

attributes :

`instance_id`	identifier of entity instance
`STEPfile_id`	identifier in STEP file
`reference_count`	number of references to instance
`application_marker`	reserved
`attributes`	list with pointers to attributes of the entity

methods :

`Name`	delivers the name of the entity
`opcode`	delivers a code for the entity
`STEPwrite`	writes entity in a STEP file
`BeginSTEPwrite`	writes all referenced entities in a STEP file
`STEPread`	reads attributes from STEP file

STEPattribute

This class represents an attribute and information about it.

attributes :

shared	is it refernced from other entities
nullable	must an attribute be used
type	type of attribute
name	name of attribute
ptr	pointer to value of attribute
type_name	name of the type in the express schema

methods :

aread	reads attribute from STEP file
screen_read	reads attribute from standard input
aprint	writes attribute in STEP file

STEPattributeList

List of attributes of the type STEPattribute.

STEPenumeration

Represents enumeration types.

2.3.3.1.3 Registry Classes

The registry is a directory of the contents of an Express schema. In contrary to the previous classes here the information about the abtract model is stored rather than the data itself stored in the model (the product information). The classes itself are not schema dependent, but they are provided with information about the schema.

The following classes are part of the Registry classes :

EntityDescriptor

Represents an entity.

attributes :

Name	name of the entity
SuperTypes	from these enities was inherited
SubTypes	these entities inherit
AttributeDescriptors	list with attributes

AttributeDescriptor

Represents an attribute.

TypeDescriptor

Represents a type.

2.3.3.1.4 Data Editor Library

This library provides functionality for the manipulation of instances or goups of instances. This includes the search of instances for certain information or the conformance checking.

The most important classes are :

InstMgr

Each instance of STEPentity has an entry in an InstMgr object. Each session has one InstMgr object.

MgrNode

Each entry in the InstMgr object has an instance of MgrNode. It holds important information about the instance.

STEPFile

Controls the handling of STEP exchange files.

CommandManager

Provides the functionality for the manipulation of entities.

DisplayNode

Each instance of an entity has an instance of DisplayNode. It contains information about the diplay status of the instance.

2.3.4 WinStep

> WinStep is a software that creates the schema specific part of the SCL (STEP Schema Class Library) from an Express schema.

The basis of the creation is ISO 10303-23 (binding of the SDAI to C++). Here happens the change from the Express model to an object-oriented class model. WinStep uses the SCL tool Fedex Plus to create the C++ classes.

For each Express schema WinStep creates the following files :

```
make_schema
schema.h
schema.cpp
Sdaiclasses.h
SdaiAll.cpp
SdaiNAME.h
SdaiNAME.cpp
SdaiNAME.init.cpp
```

Where *NAME* is the name of the Express schema.

2.3.5 Express Graphical Editor (EGE)

> The Express Graphical Editor is a tool for creating and editing Express-G diagrams. The diagrams can be transfered to Express files.

The created Express files can be used as an input for WinStep.

2.3.6 Infomodel.lib

> Infomodel is a library that controls the use of the SCL. It includes the STEP Schema Class Library and some additional classes and functions.

Infomodel was created at the *DiK* and *PTW* at the University of Technology Darmstadt. The central class is `CInfoModel`. An instance of this class represents the STEP database. It provides methods for handling the database and the necessary SCL functions.

further readings : [Schzk99]

2.3.7 AP224_IM.dll

This library provides functionality to handle the STEP database and is specialized to the datamodel used in *FESTEVAL* (adapted ISO 10303-224). It mainly uses Infolmodel.lib.
In *FESTEVAL* this is the only interface used to access the STEP database.

All operations to create, manipulate or delete instances in the STEP database and the handling of the STEP file must be initiated here. The interface to AP224_IM.dll are the global functions defined in the file `AP224_IMApi.h`. Each of these functions is calling an appropriate method of an instance of the class `AP224_IM`. Here the actual process is administrated.
The instances in the STEP database are created by the global functions `Create_*`. These functions are determining the parameters for the instances from an object of the class `CTransferObject`. The `Create_*` functions must be adapted to the datamodel.

The functions of `AP224_IMApi.h` used in this thesis are :

```
CSTEPUG* IMCreateFeature(CTransferObject *param);
CSTEPUG* IMCreateBaseShape(CTransferObject *param);
CSTEPUG* IMCreateFace(CTransferObject *param);
CSTEPUG* IMAddConstraints(int index, CTransferObject *param);
void IMClearModel();
void IMReadFile(const char* fname, int &anzahl);
void IMWriteFile(const char* fname);
void IMDeleteInstances();
```

An important class of AP224_IM is `CTransferObject`. An instance of this class can be used to store attributes of different formats. For identification, each attribute has a string identifier (`char*`). `CTransferObject`-objects are used to store the parameters of an instance in the UG database (e.g. a feature). The `CTransferObject`-object is given to the functions which are creating the instance in the STEP database (`Create_*` functions).

All manipulations in the datamodel must be considered in this library because it accesses the datamodel.

2.3.8 Visual C++

The programming language C was created in the beginning of the seventies. Its roots are mainly found in the programming language Algol (at the same time Niklaus Wirth developed the programming language Pascal from the same roots). In fig. 2-8 a brief history of different programming languages can be seen.

In the middle of the eighties, C++ resulted from the attempt to include object-oriented elements in C. C++ is a so-called hybrid programming language, because it unites procedure-oriented elements (eg. global functions) and object-oriented elements (e.g. classes, methods). Both programming techniques, object-oriented and procedure-oriented, are possible with C++.
Visual C++ is the implementation-environment for C++ programs from the company Microsoft.

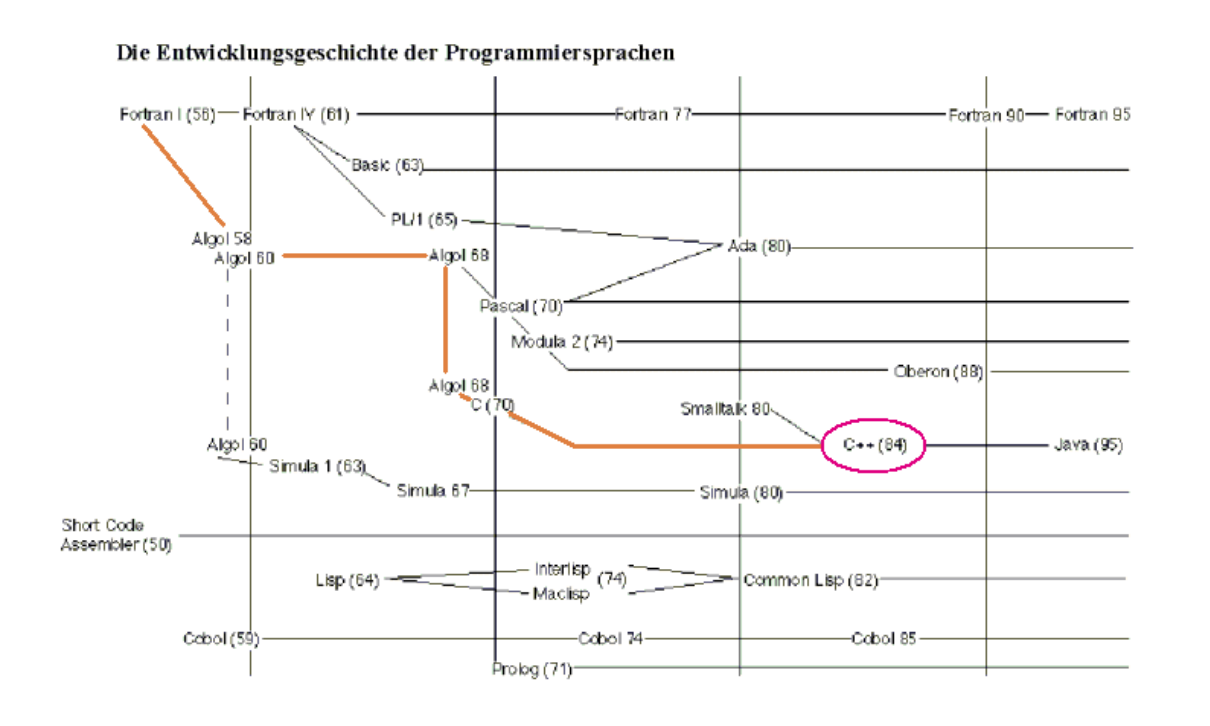


fig. 2-8 : development of programming languages *(source : [Eder2000])*

In the recent years Visual C++ became quite popular in the PC market. Besides the compiler and the linker, Visual C++ also provides some libraries, a debugger, an editor and some other tools. Visual C++ is only available for Windows based systems.

further readings : [Eckel2000], [Strou94], [Cub98]

2.3.9 UML tools / Rational Rose

The different diagram types of the UML notation can be created and manipulated with UML tools.

Most of the UML tools provide further functionality like :

- *round-trip-engineering :* Support of methodologies in all stages of software development

- *automatic code generation :* Sourcecode is created from the class diagrams in any object-oriented programming language

- *reverse engineering :* Class diagrams are created automatically from existing source code

In this thesis the UML tool *Rational Rose* from the company Rational was employed for the use-case diagrams and the class diagrams. A screenshot of Rational Rose can be seen in fig. 2-9. For other types of diagrams, *Visio Professional* was used. Visio Professional is not a special UML tool but a general template based drawing tool for all kinds of diagrams.

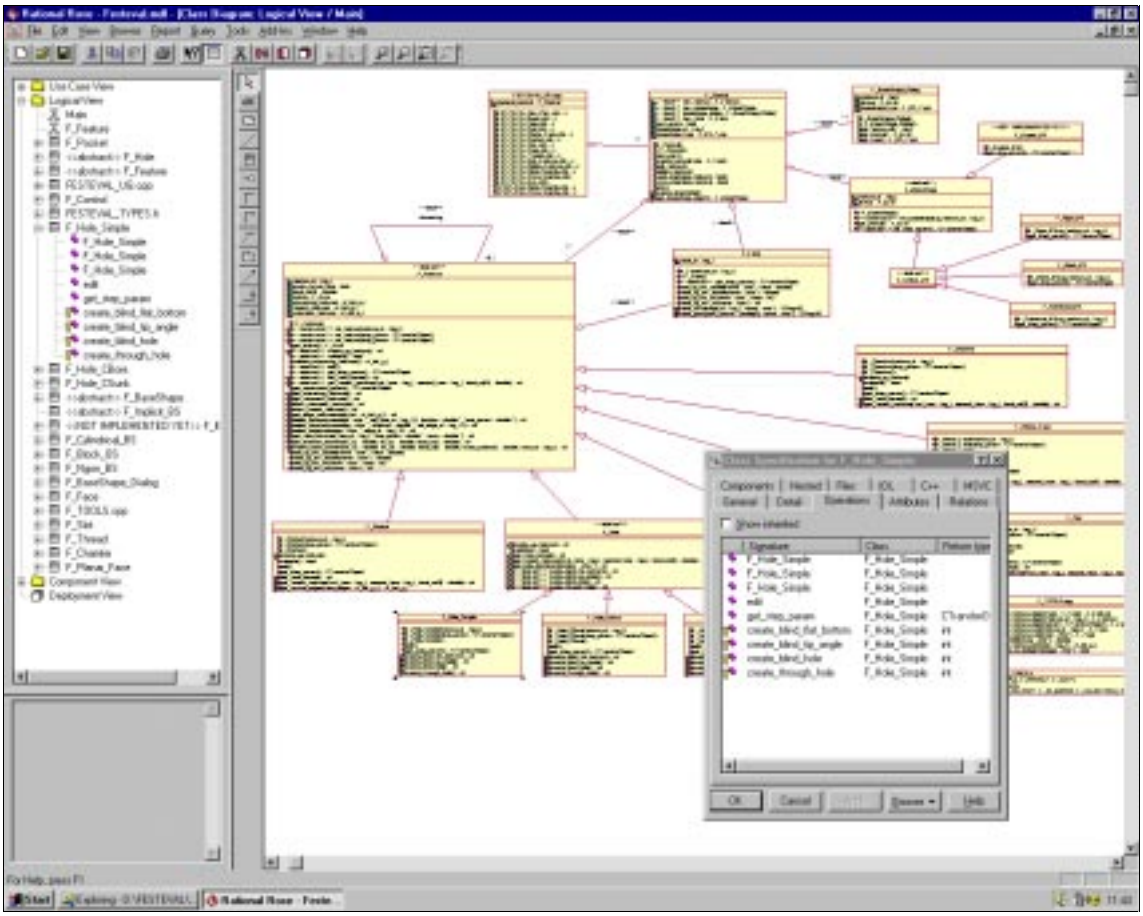


fig. 2-9 : screenshot Rational Rose

2.3.10 Tools overview

Fig. 2-10 shows the tools and how they are linked together. The tools have to be used in the following order if the STEP datamodel has to be extended :

- extension of the Express-G schema *(EGE)*
- saving of schema as Express file *(EGE)*
- generation of SDAI C++ classes from Express-schema *(WinStep)*
- replacement of old SDAI classes *(Infomodel.lib)*
- modification of the access interface *(AP224_IM.dll)*

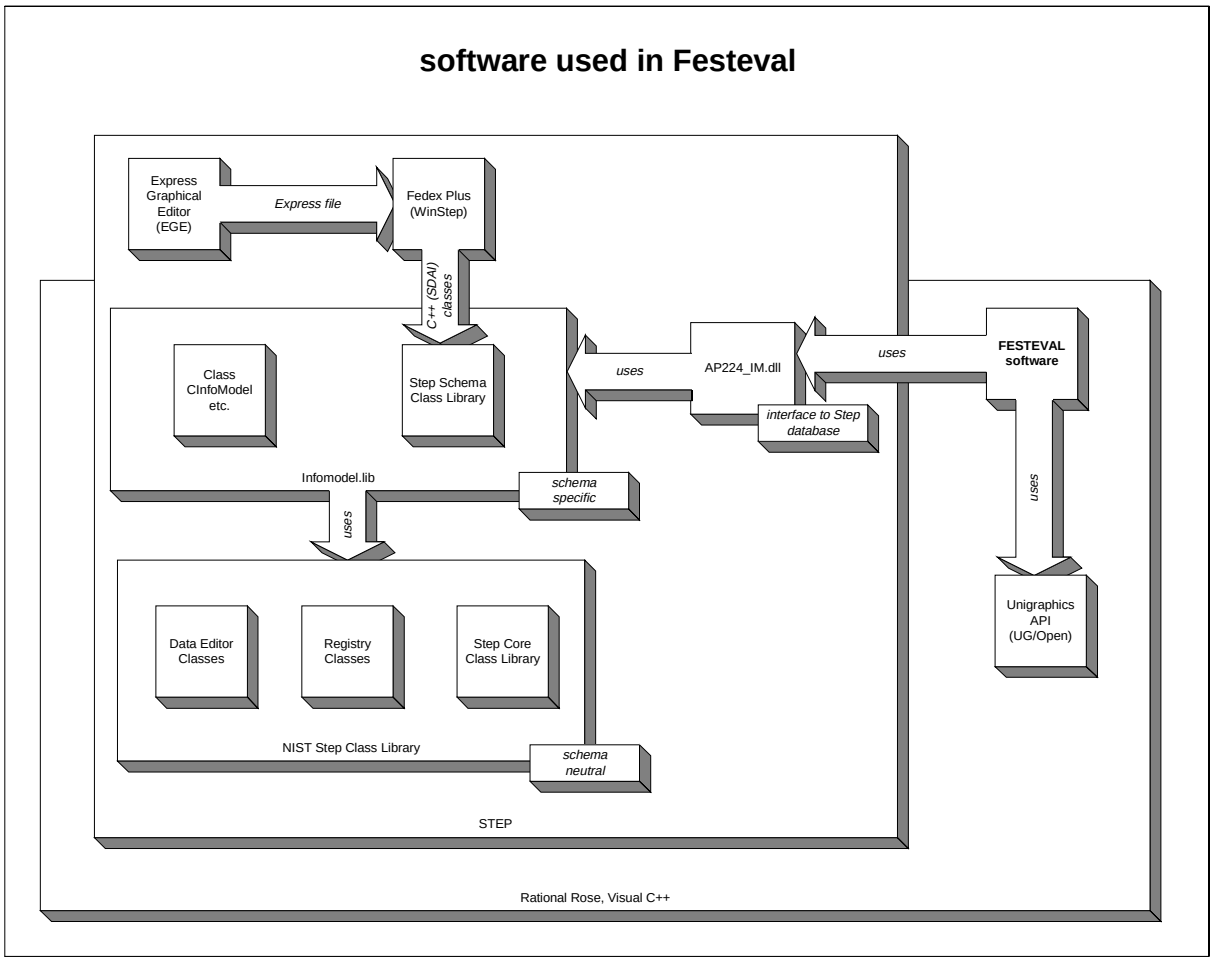


fig. 2-10 : overview of software tools

2.4 Structure of the thesis

The structure of this thesis corresponds to the stages of software development :	
- analysis	(chapter 3)
- design	(chapter 4)
- implementation	(chapter 5)
- use	(chapter 6)

The documentation is as far as possible in chronological order. Since software development is a largely dynamical and iterative process, this order can not be complied in reality. In this documentation all traverses of the development cycle are united, while in reality each stage was done several times in different traverses of the development cycle.

During the chapters dealing with the software development, the use-cases

- *new part*
- *create feature*
- *write STEP file*

are used exemplarily to document the development of the software. All further use-cases were treated similarily. The level of detail is continously rising from the first use-case diagrams up to the sourcecode and finally a running version of the program. A graphical overview of the structure of this thesis and the main focus of the chapters can be seen in fig. 2-11.

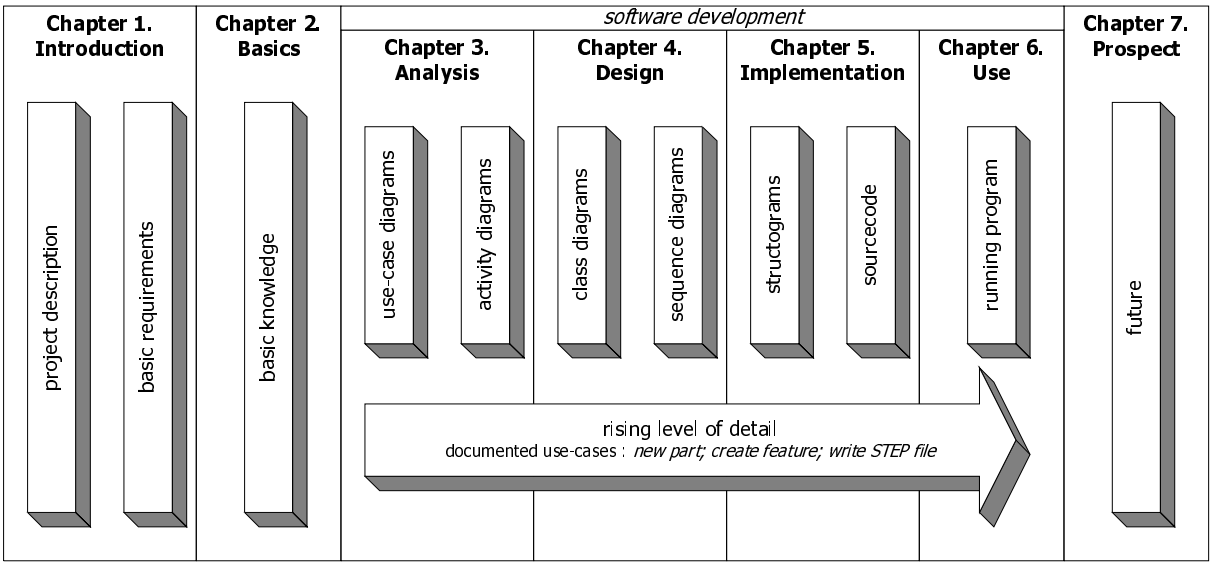


fig. 2-11 : overview of thesis

3. Analysis

In the analysis stage the sorrounding conditions of the project are analysed.
A major part is covered by the the examination of use-cases and the actions that are initiated by them.

Results of this stage are the description of requirements, mechanism, processes etc. that are important for the function of the software. This stage is very interdisciplinary, because the knowledge and the expectations of all enlisted people (users, software developers, specialists etc.) are brought together.
The results are direct inputs for the next stage, the design.

3.1 Use-cases

The examination of uses cases helps to determine the expectations and requirements that the user has about the software. Internal processes are irrelevant at this point, if a user is not directly affected by them.

In the use-case diagram in UML notation each interaction between a user and the software is called a use-case. The notation is an oval circle with the description of the use-case. The actors are represented as stickmen. The connection between actors and use-cases is represented by arrows. The initiation of a use-case by another use-case is indicated by an arrow between them. A user is normaly using the software via the GUI (graphical user interface). Thus use-cases are helpful to plan the functionality the GUI has to provide (buttons, menus etc.).
Actors must not necessarily be real users. In use-case diagrams also other systems which are communicating with the software can be described as actors. In this thesis also the used databases (the Unigraphics database and the STEP database) are shown as actors in the use-case diagrams. Thereby also the expectations of the users about the databases are represented. Here, for example, the use-case diagrams are allready making shure that a feature that was created by the user is validated and stored in the UG database. How this is done is not part of the use-case diagrams and will be examined later. Only the requirements from the user are determined.
In this thesis the examination of the three use-cases

- *new part*
- *create feature*
- *write STEP file*

will be documented.

3.1.1 New part

The use case *new part* starts a new session in the FESTEVAL environment. To perform this, a part that consists at least of a baseshape must exists in the UG database. After the user has pressed the button *new part*, he has to choose a baseshape from the part. After this, all features related to the choosen baseshape are automatically validated. The user can choose if he wants to have the invalid features deleted. If he does not agree, the part can not be used in FESTEVAL and the session will not be started.

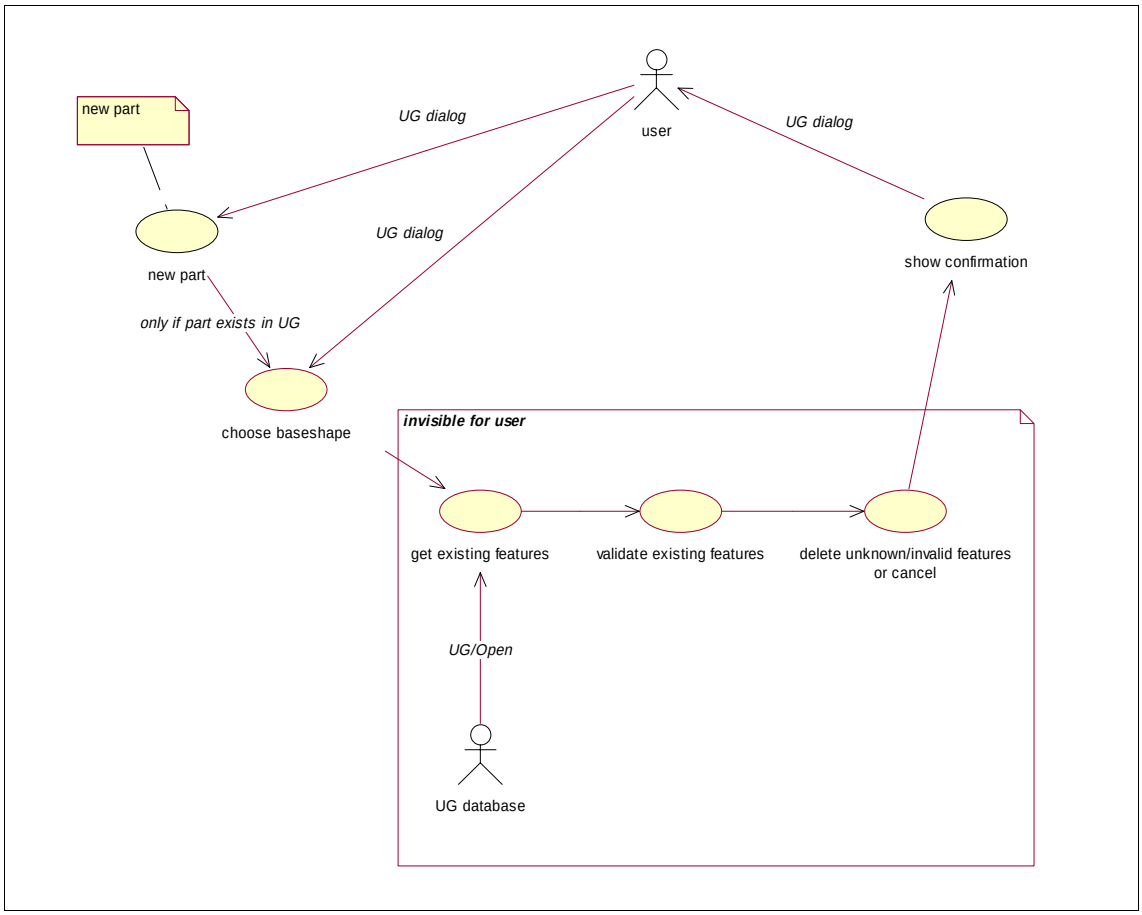


fig. 3-1 : use-case diagram new part

3.1.2 Create feature

After a session is started in the FESTEVAL environment, new features can be created in the opened part. After pressing the button related to the type of feature, the user has to enter all the necessary geometric parameters. The feature is then created in the UG database and will be validated after the creation. An invalid feature will be deleted again automatically. This ensures that only valid features are used in the FESTEVAL part.

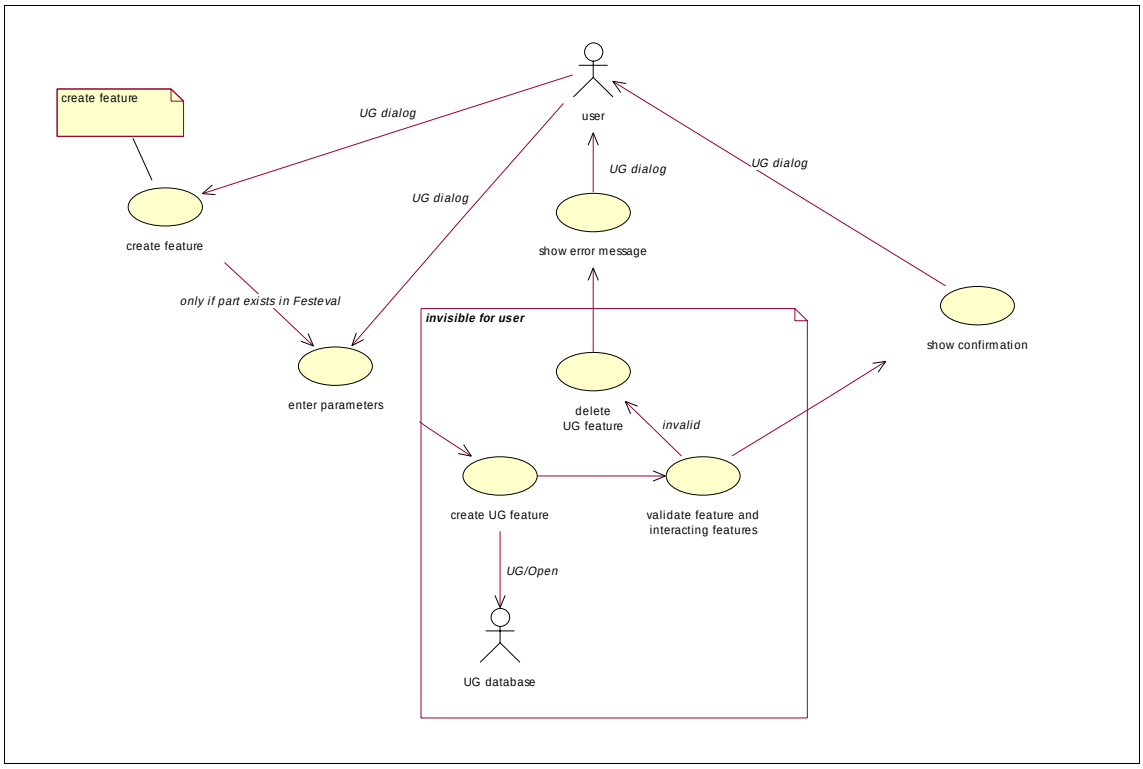


fig. 3-2 : use-case diagram create feature

3.1.3 Write STEP file

After pressing the button for writing the STEP file, the user has to choose the directory and filename for the new file. To create the STEP file, a STEP conform database has to be used. If not existent at this time, it has to be created before the STEP file can be created (see chapter 3.2.4 and 3.2.5 for a more detailed analysis of this operation).

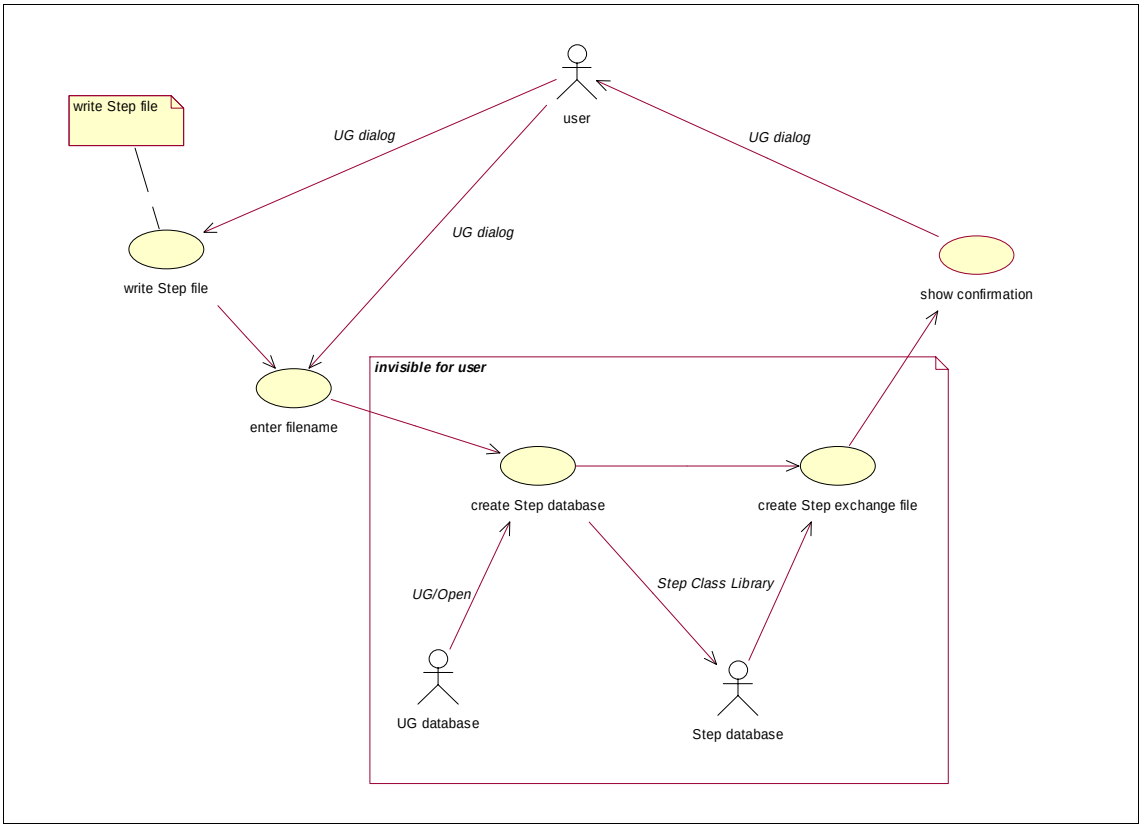


fig. 3-3 : use-case diagram write STEP file

3.2 Internal processes

In this section the internal processes (invisible for the user) that are initialised by the use-cases are examined.

This includes for example the way of storing the data or the Steps that are necessary to create a physical file.

3.2.1 Creation of geometry in Unigraphics

Part of the FESTEVAL design environment is the creation of features. The purpose of providing new functions is to include a validation during the creation process. This enables the designer to react directly on the result.

The functionality for creating features consists of user dialogs for determining the necessary information and of the creation of the feature in the UG database.

3.2.2 Validation of features

In FESTEVAL each feature must be validated. Only if the validation is succesful the feature is allowed to be used in a FESTEVAL part. The idea behind this validation is to have a view to the next stages of the process chain already in this early stage. There are many different ways to design the same geometry. Although if the difference is not seen in the geometry, it is stored in the database and will be read and used by the next stages of the process chain. The semantics of the product data has to follow certain rules, otherwise it will be misinterpreted. The validation makes sure that these rules are not disobeyed during the design.

Parts of the validation are for example :

- are the tools requred to manufacture the feature available ?
- are certain geometric requirements fulfilled ?
- is the type and arrangement of the feature sensible ?

The way the validation works is different for each type of feature. Hence each feature type has to have an own method for validation. An example for the validation is shown in fig. 3-4. In the cases shown on the right, the feature is invalid because a pocket is used to define a slot or a STEP. This figure also shows how a valid feature can become invalid by modification.

The whole subject of validation is handled by the SCPM and is therefore not part of this thesis. Part of this thesis is the integration of the methods provided by the SCPM into the design.

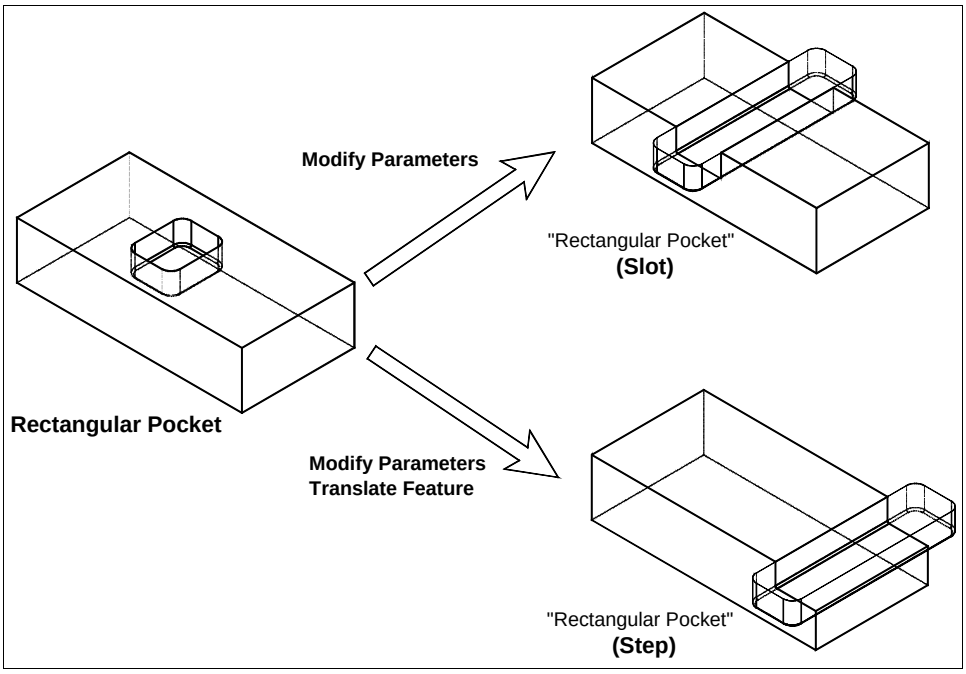


fig. 3-4 : examples for invalid features (source : SCPM)

3.2.3 Determination of interdependencies

There are different kinds of technological and geometrical interdependencies that can exist between features. A simple example is a volume interaction, in which two features are intersecting in a certain area. Information about such interactions can be useful for tool machines. By an interpretation of the interactions the control system for the tool machine can automatically determine an optimized manufacturing procedure. This circumstance is one of the main aspects of FESTEVAL. Therefore it must be possible for each feature to determine its interdependencies and to store them in the STEP database.
In the datamodel used in FESTEVAL the interdependencies and further information about the manufacturing procedure are stored in the attribute `constraints` for each manufacturing feature. Fig. 3-6 shows the Express-G schema of `constraints` in the FESTEVAL datamodel. In this thesis the interactions `volume_interaction` and `face_interaction` are treated.
A simple example for a slot interacting with a pocket can be seen in fig. 3-5.

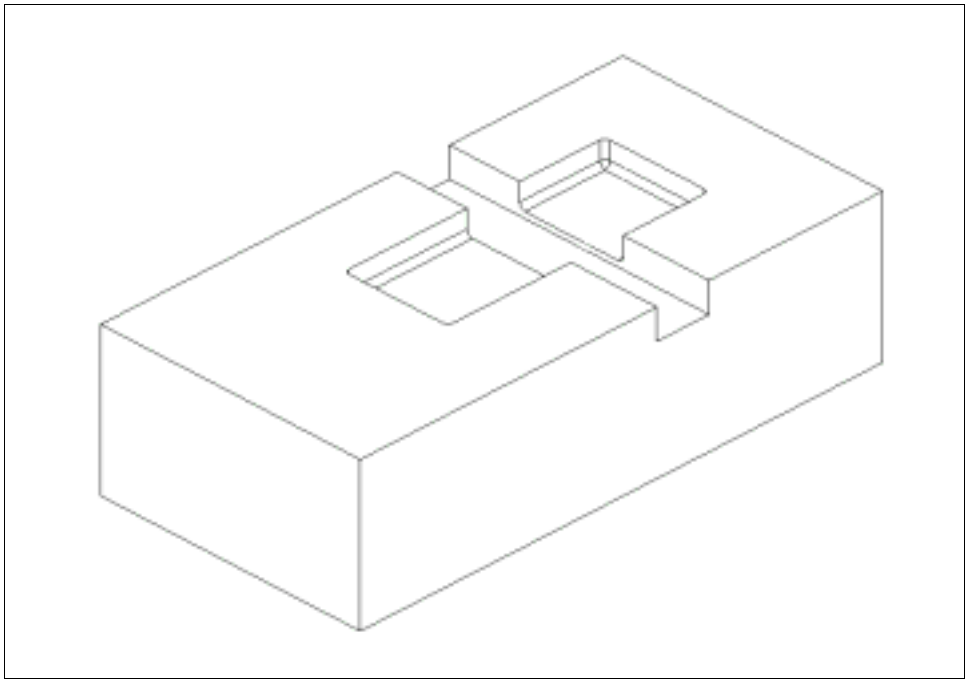


fig. 3-5 : example for interacting features

The determination of the interdependencies is handled by the SCPM and is therefore not part of this thesis. Part of the thesis is the integration of the methods into the design and the convertion of these information to the STEP database.

further readings : [Schtz98], [Schtz97]

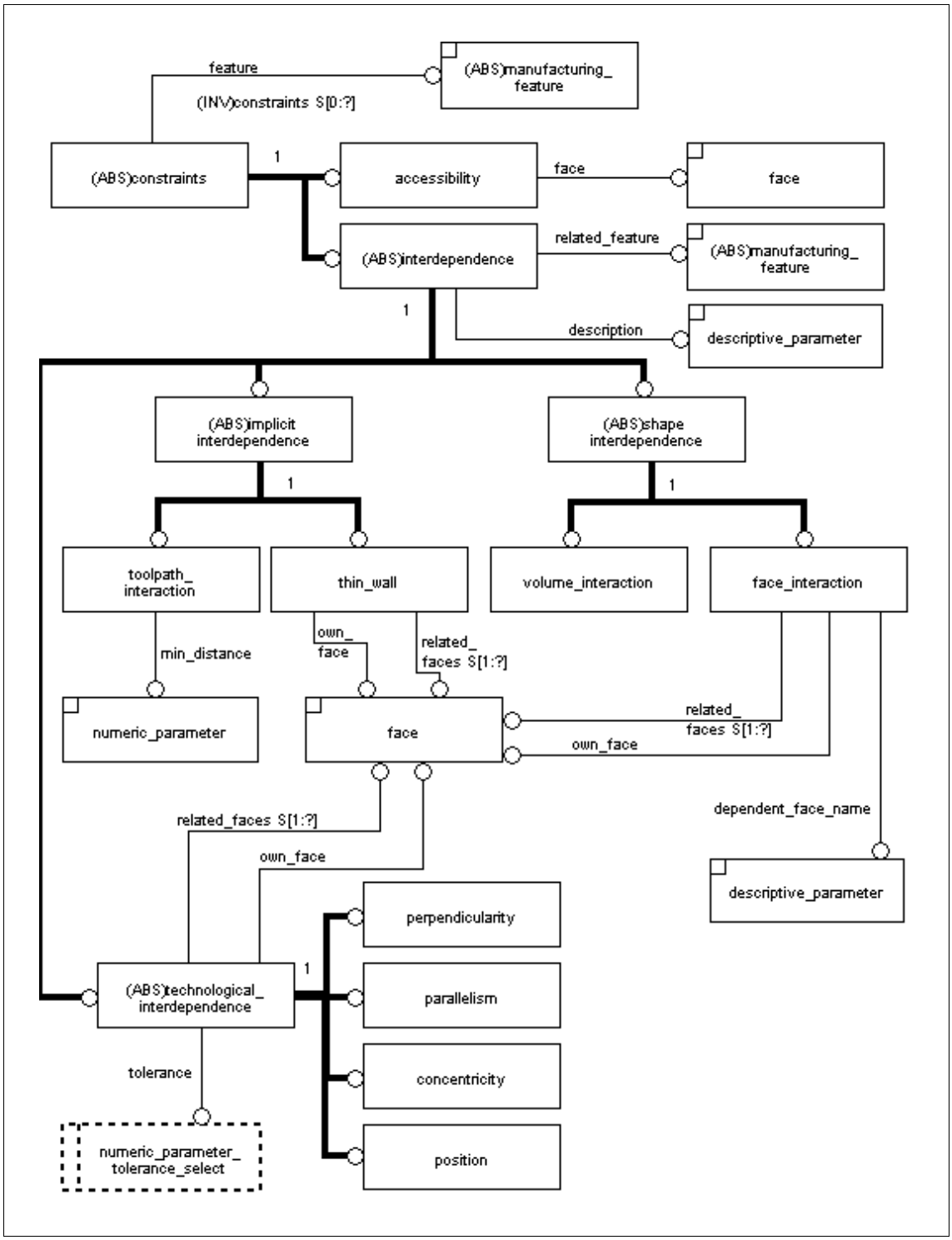


fig. 3-6 : Express-G (G.33) – constraints *(source : DiK)*

3.2.4 Storage of the *FESTEVAL* specific attributes

The datamodel used in *FESTEVAL* is an adapted STEP application protocol 224. This datamodel defines some attributes that are not part of the UG datamodel, like the interdependencies. After a feature is created in the *FESTEVAL* environment, only the attributes that are part of the UG datamodel can be stored in the UG database.
A way has to be found to prevent this information to get lost. They must be stored somewhere to be available at a later time to be stored in the STEP file. There a different approaches, and the most suitable of them must be found.

The three approaches

a) redundant databases
b) storage in the UG database
c) storage in further database

are examined in the following sections.

3.2.4.1 Redundant databases

A solution could be to build and maintain the STEP database and the UG database at the same time. All attributes would be stored in the STEP database and be available at any time.
The disadvantage is the effort to maintain two partly redundant databases. Each modification in the UG database has to be made immediately in the STEP database as well. Besides the creation of features this includes also the deletion and editating of features. The way the functions of *FESTEVAL* are accessing the databases in this solution are shown in fig. 3-7.

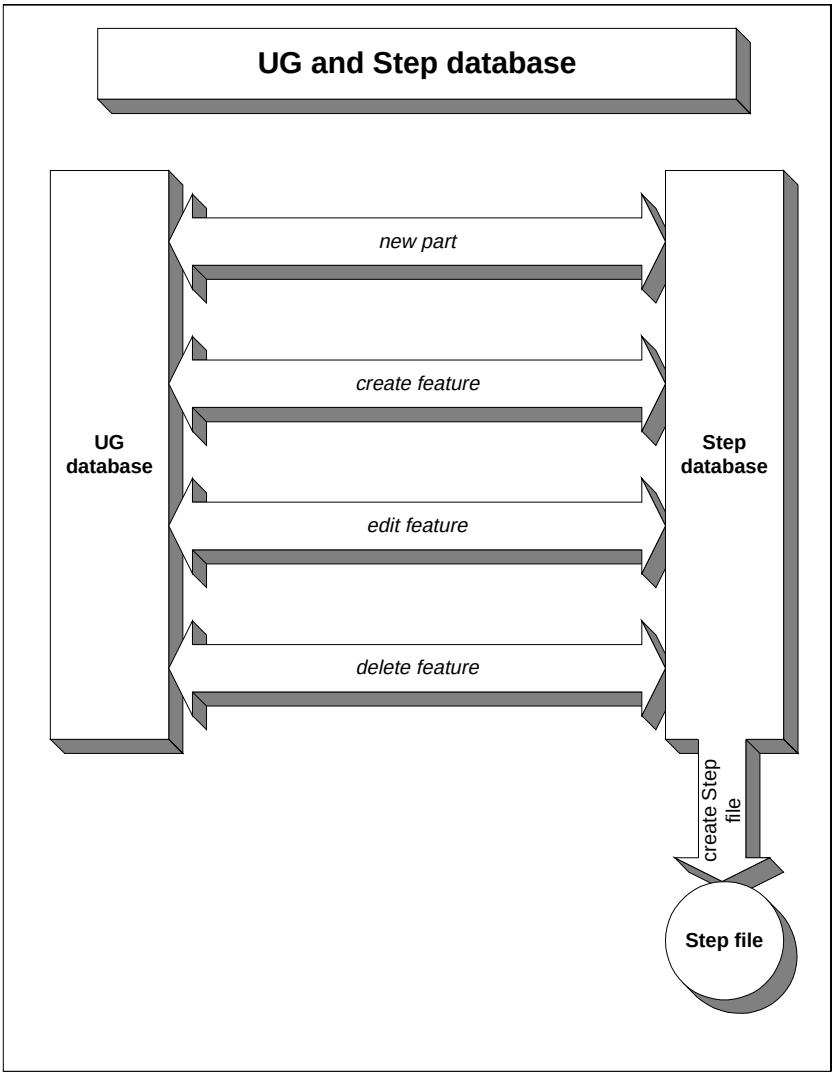


fig. 3-7 : storage in redundant databases

3.2.4.2 Storage in the UG database

To avoid the redundancy it must be attempted to store all required information in only one database. This could only be the UG database because it is absolutely necessary to work in Unigraphics. Another attempt is therefore to extend the UG datamodel by the required additional attributes to store all important information in the UG database. In this solution the STEP database is created only immediately before the creation of the STEP exchange file and is deleted directly afterwards.
The disadvantages of the first attempt are not occuring in this solution. The problem here is to extend the the UG datamodel by the required attributes. This solution is only possible if UG/Open provides enough functionality for extending the datamodel. Fig. 3-8 shows how the functions of *FESTEVAL* would access the databases in this solution.

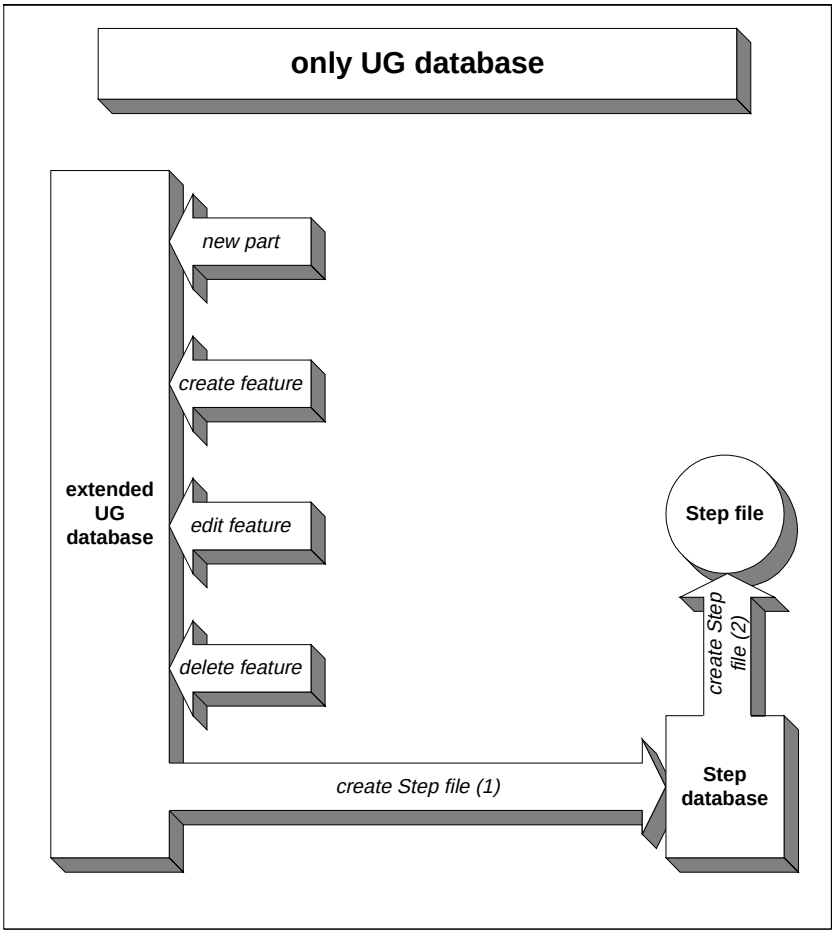


fig. 3-8 : storage only in the UG database

3.2.4.3 Storage in further database

Another attempt to avoid the redundancy is to store the additional attributes in a further database. This database could be a simple spreadsheet with a reference to the feature in the UG database and the additional attributes. The difference to the previous attempt is the creation of a new datamodel instead of extending the existing one.
The disadvantage is the effort of creating and maintaining an additional database. Fig. 3-9 shows the access to the databases iin this solution.

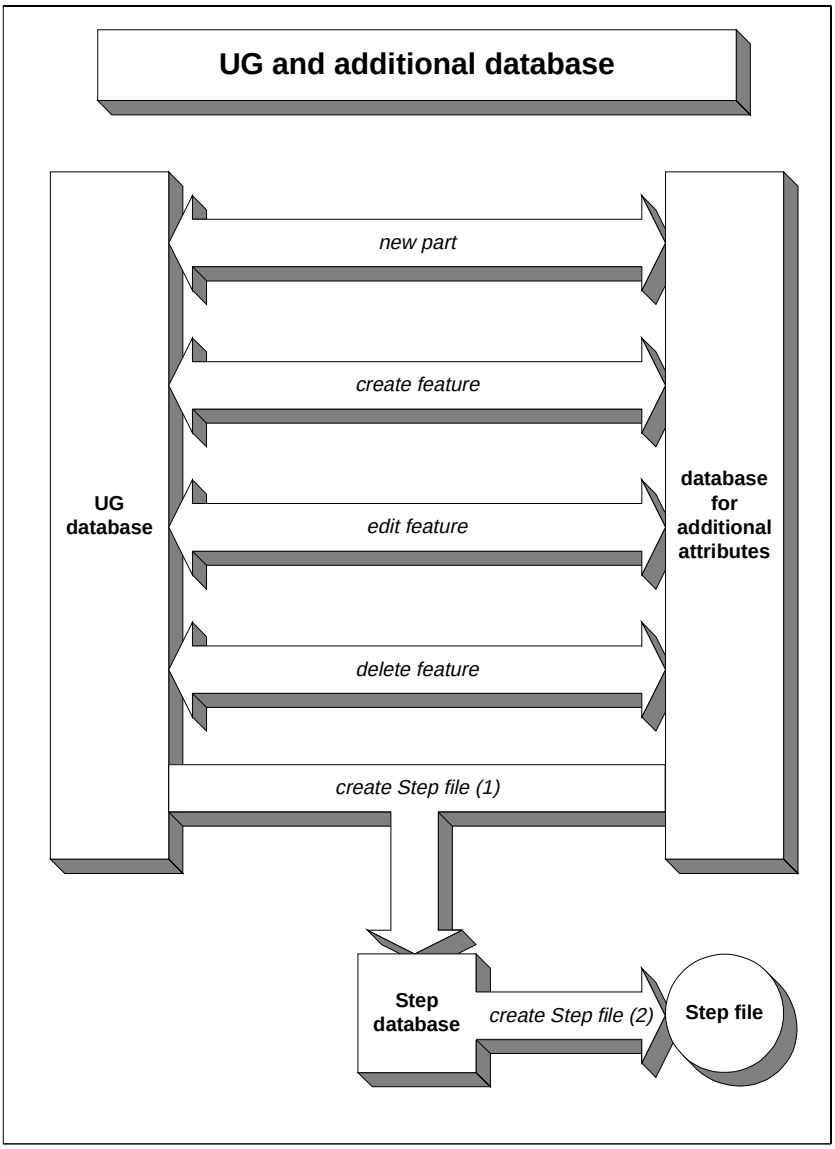


fig. 3-9 : storage in further database

3.2.4.4 Review and decision

A major issue for making a decision for one of the solutions is the potential of the software tools to support the attempts.

a) The first solution will cause big efforts for maintaining the STEP database. While it is easy to create data in that database, the modification of data is very complicated. To delete a feature, for example, it is necessary to delete all the attributes connected to it, but not the ones which are also referenced by other instances in the database.

b) The second solution is easy and straight forward, but it requires extensive extensions in the UG datamodel. UG/Open provides enough functionality to store additional parameters of

simple datatypes with each feature. This could be used to easily store additional attributes like dimensional tolerances.
A difficulty is, as well as in all the other solutions, that the independencies in a feature can change, even if this feature was not modified. This happens when interacting features are modified. Although the interdependencies could be stored in additional UG attributes, the changes to them would have to be modified whenever necessary.
A solution to avoid this effort is to determine the interdependencies only directly before they are needed (to create the STEP database and STEP file). This ensures the use only of the actual state of the interdependencies and the effort to maintain the entries in the database is avoided.

c) The third solution is technically no problem, but unnecessary on account of the easier possibilities of the second solution.

The second solution is irreproachably the best one and will be used in the further progress of the thesis. It allows either to store all necessary information in the UG datamodel or, where possible, to determine the actual information when needed (e.g. the interdependencies to create the STEP database).

3.2.5 Creation of STEP file

The decision made in the previous section results in creating the STEP database directly before the creation of the STEP file. After the STEP file was created the STEP database can be deleted again.

Hence, the following Steps are necessary to create the STEP file :

1. determination of the file name (user input)
2. creation of a STEP database
3. determination of the parameters for the baseshape from the UG database
4. storage of the parameters for the baseshape in the STEP database

for all features :
5. determination of the parameters for the feature from the UG database
6. storage of the parameters for the feature in the STEP database
7. determination of the interdependencies for the feature
9. storage of the interdependencies for the feature in the STEP database

10. creation of the STEP file from the STEP database
11. deletion of the STEP database

For the creation of the features or the baseshape in the STEP database, the parameters have to be determined from the UG database and be stored in a format that is suitable for the creation in the STEP database.

The software should also provide functionality to create STEP files outside of the FESTEVAL design environment. Thereby the same modules of the software are used inside FESTEVAL to provide a STEP processor that is independent of the FESTEVAL environment.

3.3 Activities

Activity diagrams are used to show activities, order of processes, parallel processes and resonsibilities. They can be universally employed in different stages and for different purposes. In the analysis stage, they are very suitable to describe and detail the expected processes and their affiliation to certain functional groups.

> In this thesis activity diagrams are shown for all documented use-cases. Contradictory to the use-case diagrams, they are also visualizing the resulting internal processes.

On a certain level of detail the processes are shown as black boxes. Further detailing is not part of the analysis stage and will be done in the design and implementation stage.
The activity diagrams are showing very clearly the requirements and the rough realisation. Hence they are important results from the analysis stage and an important input to the design stage.

In activity diagrams, each activity is represented as an oval. The activities are connected by arrows, which represent the chronological order of the activities. The rhombuses are representing decision. The activities can follow only one way out of a decision. The black dot initialises the use-case, the encircled dots are representing terminations.

3.3.1 New part

See fig. 3-10. The following activities are of importance :

check if part exists : The use case *new part* can only be accomplished if a part (at least the baseshape) exists in the UG database.

dialog to choose baseshape : Of all the features of the actual UG part the user has to choose one as a baseshape for the FESTEVAL part.

check if choosen baseshape is ok : The choosen baseshape has to be a UG feature that can be interpreted by FESTEVAL as a baseshape. Otherwise it is not possible to use it in FESTEVAL.

get existing features : All features existing in UG that are related to the choosen baseshape have to be determined.

validate existing features : The features can only be used in FESTEVAL if they are satisfying the FESTEVAL validation. The invalid features have to be determined.

dialog delete/cancel : The user has to choose if he wants to delete the invalid features or if he wants to cancel the operation.

part_exists = true : This means that by now a valid part exists in the FESTEVAL environment. A session is started in FESTEVAL.

3.3.2 Create feature

See fig. 3-11. The following activities are of importance :

check if part_exists : This ensures that a valid part exists in the FESTEVAL environment and that a session is started. Otherwise no proceeding is possible.

dialog for paramters : The user has to enter all the geometrical parameters that are necessary to create the feature.

create new feature : The feature is created in the UG database.

validate new feature : The newly created feature is validated.

delete new feature : The newly created feature is deleted from the UG database. Invalid features can not be used in the FESTEVAL environment.

get interacting features : The features having some kind of interaction with the newly created feature are determined

check interacting features : All the interacting features are validated. They could have lost their validity by the interaction with the newly created feature.

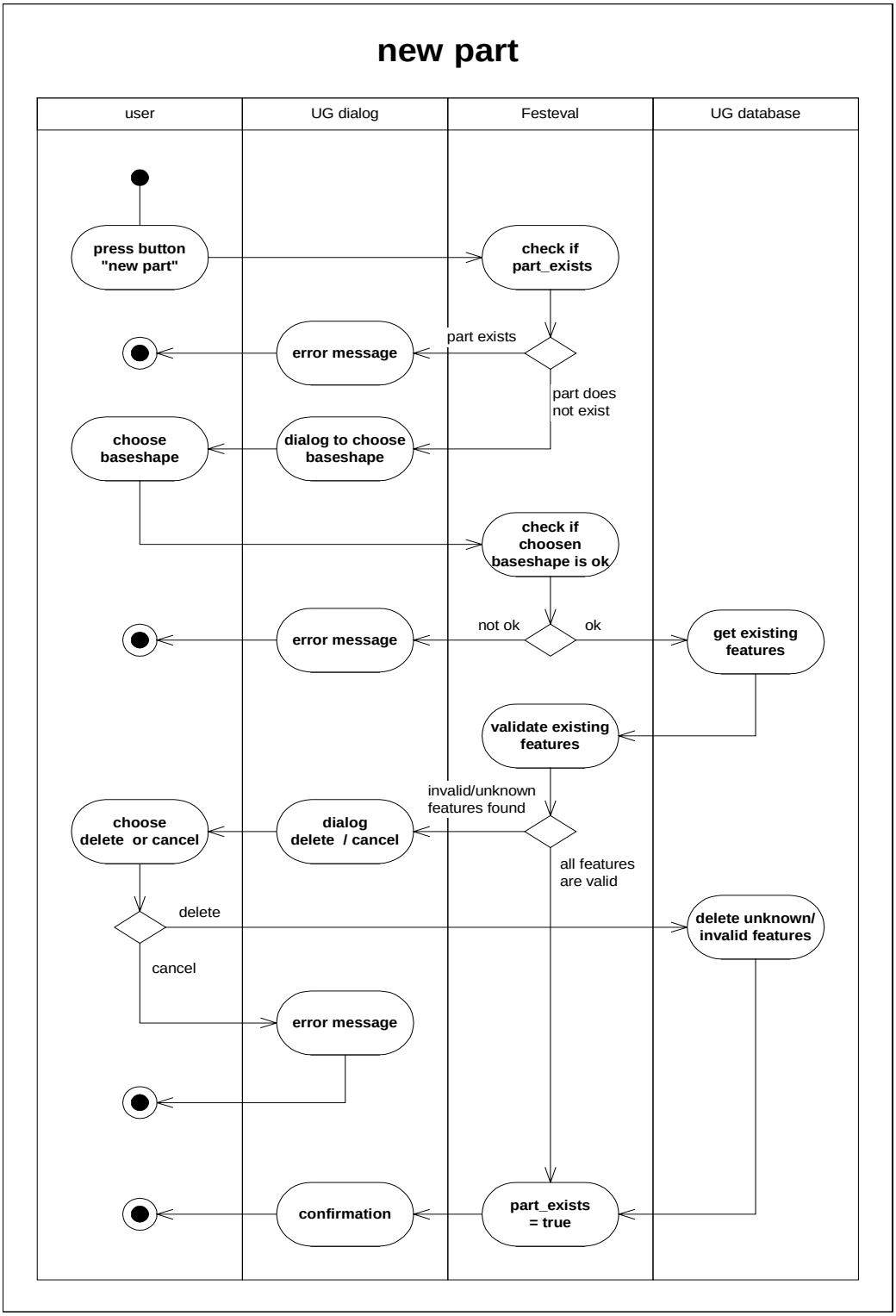


fig. 3-10 : activity diagram new part

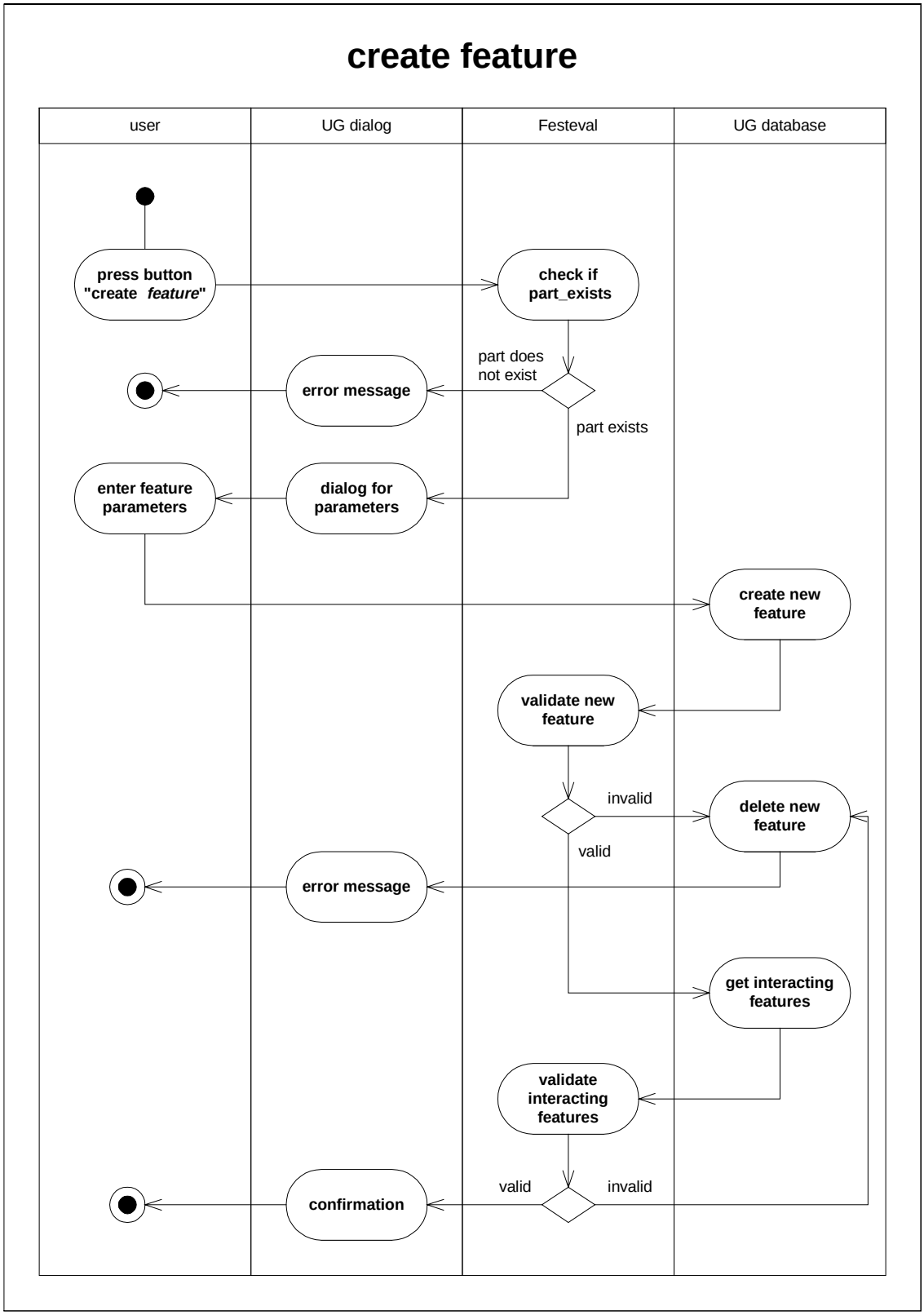


fig. 3-11 : activity diagram create feature

3.3.3 Write STEP file

See fig. 3-12. The following activities are of importance :

check if part_exists : this ensures that a valid part exists in the FESTEVAL environment and that a session is started. Otherwise no proceeding is possible.

dialog for filename : The user has to choose a name and location for the STEP file.

get baseshape : The baseshape and its parameters have to be determined from the UG database.

create baseshape : The baseshape has to be created in the STEP database.

get features : All features belonging to the FESTEVAL part and their parameters have to be determined.

create features and faces : The features and their faces have to be created in the STEP database.

get constraints : The constraints (interdependencies) of all features have to be determined.

create constraints : The constraints have to be created in the STEP database. This has to be done after all the features were created in the STEP database, because they might be referenced by the constraints (interdependencies).

create STEP file : The physical file is created.

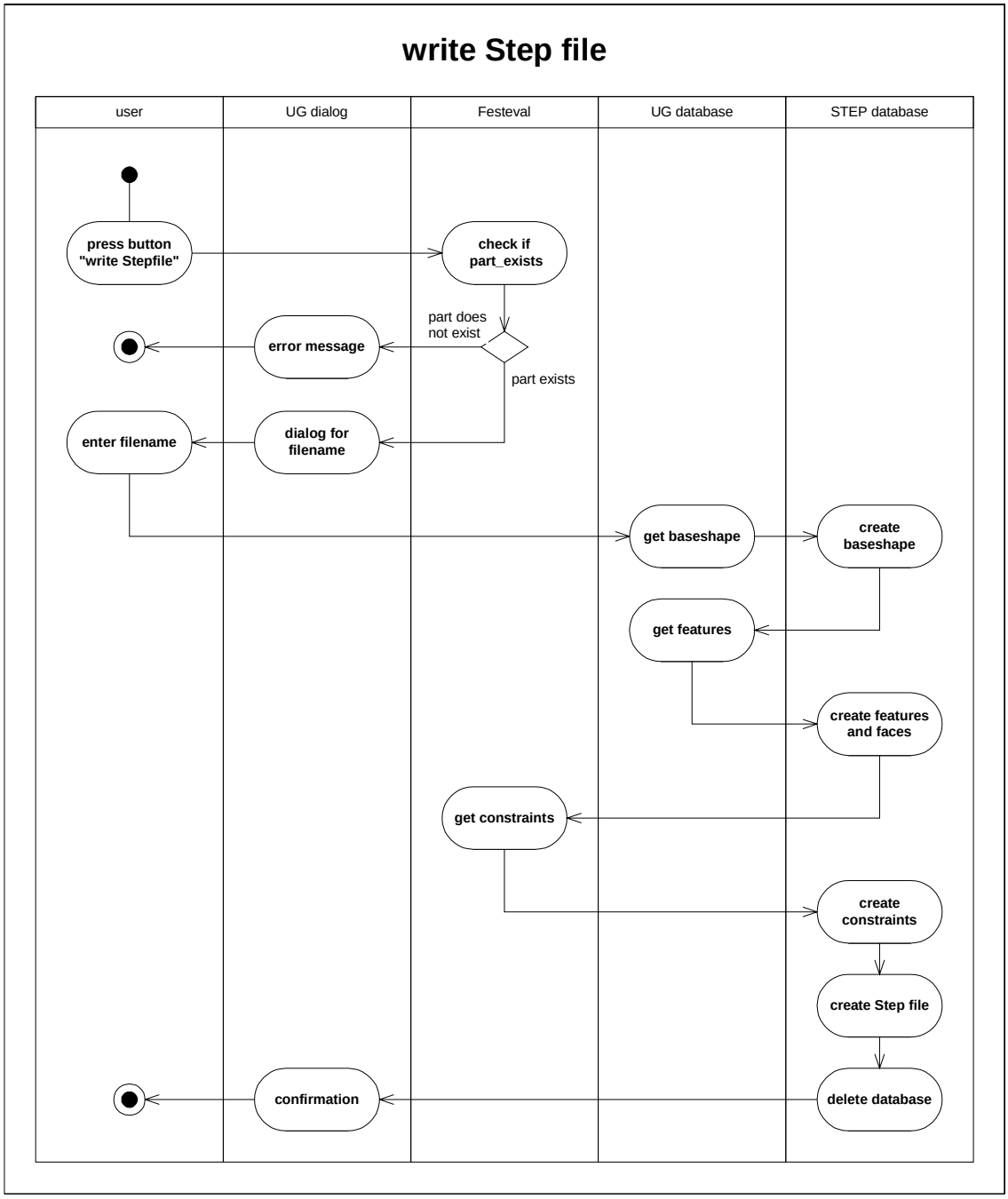


fig. 3-12 : activity diagram write STEP file

4. Design

> In the design stage, the expressions and conditions of the real world are transferred into an abstract model. The design stage is the most productive stage of the software development in view to the final result (the software).

The results of the analysis stage are the basis for the determination of the constructs needed for the static model.
The dynamic model is helpful to check if the static model has enough functionality to handle all the required use-cases.

Some notations were used in the class diagrams, which are not defined in the UML :

a) Attributes which are not classmembers but existent only locally in certain methods are represented as aggregations with the stereotype `<<local>>`.

b) Procedure-oriented parts are represented as classes with the stereotype `<<globals>>`. The class name is the name of the file, the functions are represented as methods.

4.1 Static model

> The static model describes the structure that is created by the classes and their relations among themselves.
>
> The static model is represented with class diagrams in UML notation.

In this documentation the model is divided into the part models features, baseshapes and control.

4.1.1 Features

For the representation of the features in the model an abstract class `F_Feature` is provided. This class defines the classmembers that are necessary for all types of features. The methods that work the same for all types of features are implemented in `F_Feature`, the methods that work different for different types of features are defined as abstract methods in `F_Feature`. For each type of feature a concrete subclass of `F_Feature` is provided for. This is where the feature specific methods are implemented. The class definition of `F_Feature` can be seen in fig. 4-1.

Whenever a task about a specific feature has to be performed, an object of the sub-class which corresponds to the type of the feature has to be instanciated. This object is linked to an entry in the UG database by the class attribute `feature_id` of the type `tag_t`. Each method that is called within that object is operating on the referenced feature in the UG database.

The following tasks have to be handled by the instances of these classes :

- creation of a new feature in Unigraphics
- validation of the feature
- determination of interdependencies
- validation of interdependent features
- determination of the parameters to create the STEP instance

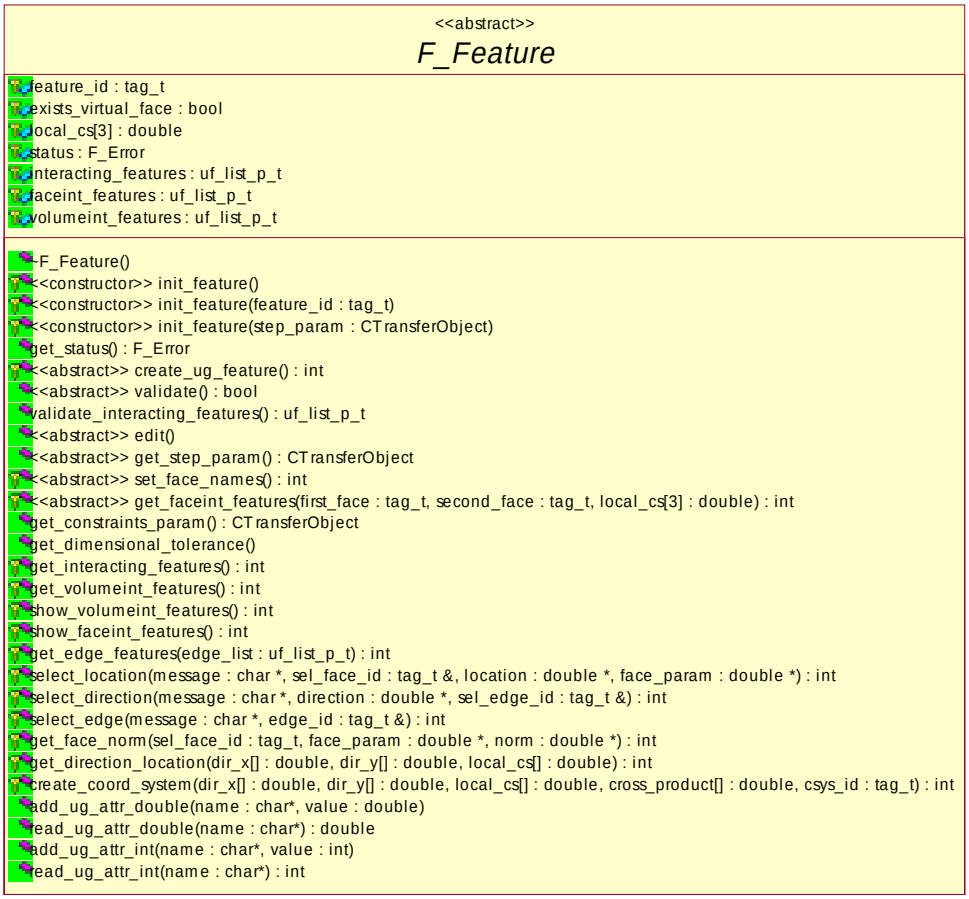


fig. 4-1 : abstract class F_Feature

There are three different starting conditions for instanciating objects of these classes. For each condition a different initialisation method, which controls the necessary processes, is provided in the super-class *F_Feature*. These initialisation methods are called from the constructors of the concrete sub-classes.

a) The feature does not exist yet. This is the case when the user wants to create a completely new feature, thus when he chooses *create feature* in the environment.

- initialisation method : `init_feature();`

Example : The user wants to create a new pocket. An instance of the class `F_Pocket` *has to be created using :* `F_Feature* pocket_obj = new F_Pocket();`. *The constructor automatically calls the method* `init_feature()` *of the class* `F_Feature`*. This method initialises the steps necessary to create a new pocket (e.g. user dialog for parameters, validation etc.).*

b) The feature exists in the UG database. Operations have to be done with an existing feature if an feature created outside of FESTEVAL has to be validated or when the existing features have to be transferred to the STEP database.

- initialisation method : `init_feature(tag_t feature_id);`

Example : An existing pocket has to be transferred to the STEP database. An instance of the class `F_Pocket` *has to be created using :* `F_Feature* pocket_obj = new F_Pocket(feature_id);` *where* `feature_id` *is the* `tag_t`*-identifier for the existing pocket. The constructor automatically calls the method* `init_feature(CTransferObject* step_param)` *of the class* `F_Feature`*.*
Now the method `pocket_obj->get_step_param()` *can be called to determine an* `CTransferObject`*-object with the parameters of the pocket. This object can be used to create the pocket in the STEP database via the* `AP224_IM.dll`*.*

c) The feature exists in the STEP database. This case occurs if a STEP file was read and the features have now to be created in the UG database.

- initialisation method : `init_feature(CtransferObject* step_param);`

Example : An pocket existing in a STEP database has to be transferred to FESTEVAL, and thereby created in the UG database. An instance of the class `F_Pocket` *has to be created using :* `F_Feature* pocket_obj = new F_Pocket(step_param);` *where* `step_param` *is a* `CTransferObject` *object containing the parameters of the feature from the STEP database. The constructor automatically calls the method* `init_feature(CTransferObject* step_param)` *of the class* `F_Feature`*. This method initialises the Steps necessary to create the pocket in FESTEVAL (e.g. creation in the UG database, validation etc.).*
Note : The methods `init_feature(CTransferObject* step_param)` *were not completely implemented during this thesis.*

The methods `get_step_param()` and `get_constraints_param()` are determining the parameters of the feature from the UG database and are wrapping them in an object of the class `CTransferObject`. This object can be used to create the feature or its constraints in the STEP database with the AP224_IM.dll. The sub-classes for the different types of features are implemented according to the super-class. Further generalizations can occur here, like from the abstract class *`F_Hole`* to the concrete sub-classes `F_Hole_Simple`, `F_Hole_CBore` and `F_Hole_CSunk`. Common methods for all kinds of holes are implemented in *`F_Hole`*.

The class `F_Face` can be used to determine the parameters of material faces wrapped in a `CTransferObject` object, which can be used to create the faces in the STEP database. The utilisation is analogous to case b) described for features above. Notice that `F_Face` is not a sub-class but an attribute of *`F_Feature`*.

The class structure with the abstract class *`F_Feature`*, its concrete sub-classes and the class `F_Face` can be seen in fig. 4-2. Also refere to fig. 2-5.

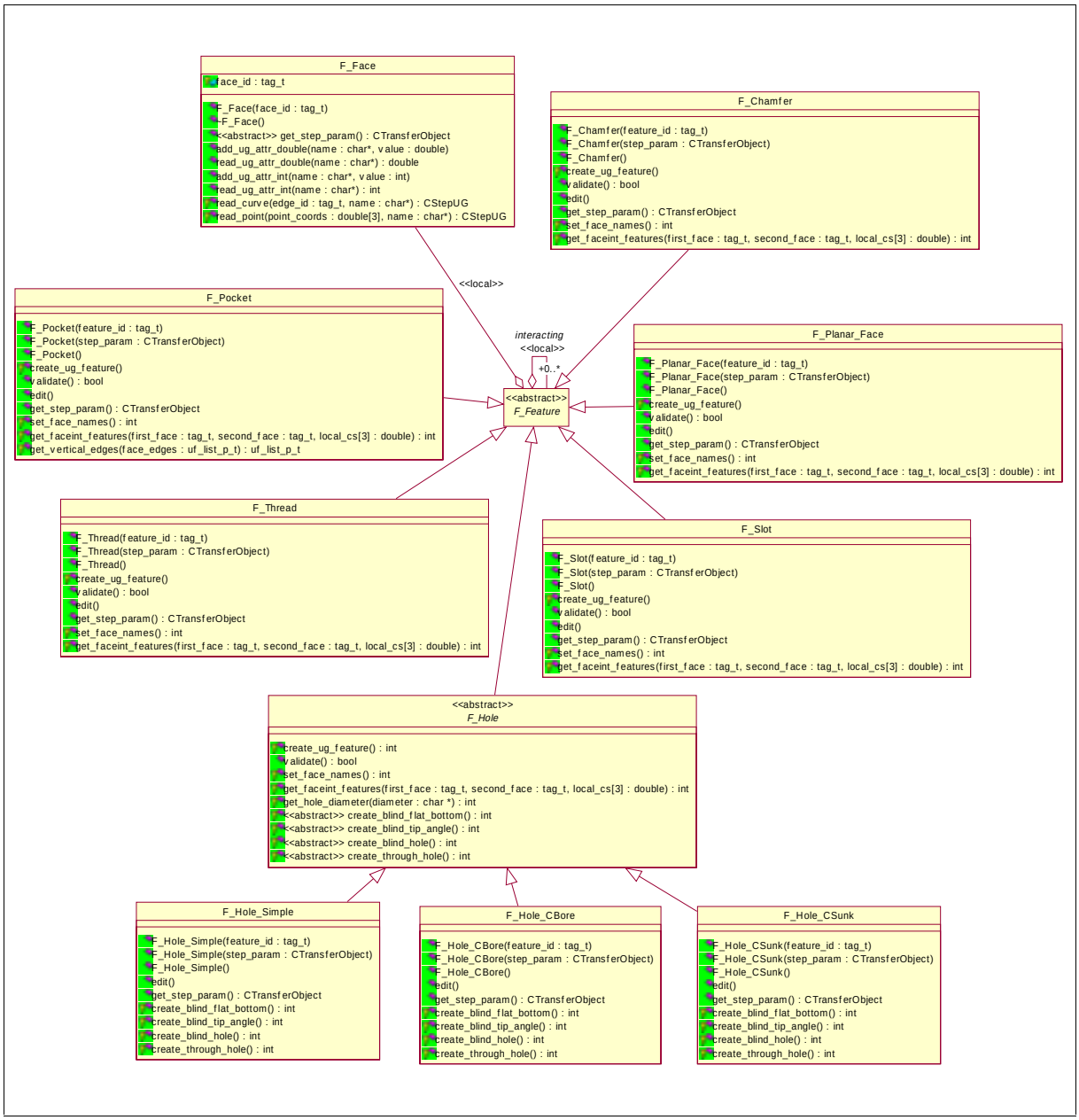


fig. 4-2 : static model for features

4.1.2 Baseshapes

In the *FESTEVAL* datamodel each part has to have a baseshape. Hence, to use a part in *FESTEVAL*, a baseshape has to be assigned to it. Like with the features there are also different kinds of baseshapes. The class hierarchy starts again with an abstract super-class *F_BaseShape*.

In this thesis only implicit types of baseshapes are implemented. Admittedly the model is already prepared for explicit types of baseshapes by the classes *F_Implicit_BS* and F_Explicit_BS.

To instanciate an baseshape object the corresponding feature in the UG database has to be stated. Thereby, the use is analogous to case b) described for features.
The class structure of the classes handling the baseshapes can be seen in fig. 4-3.

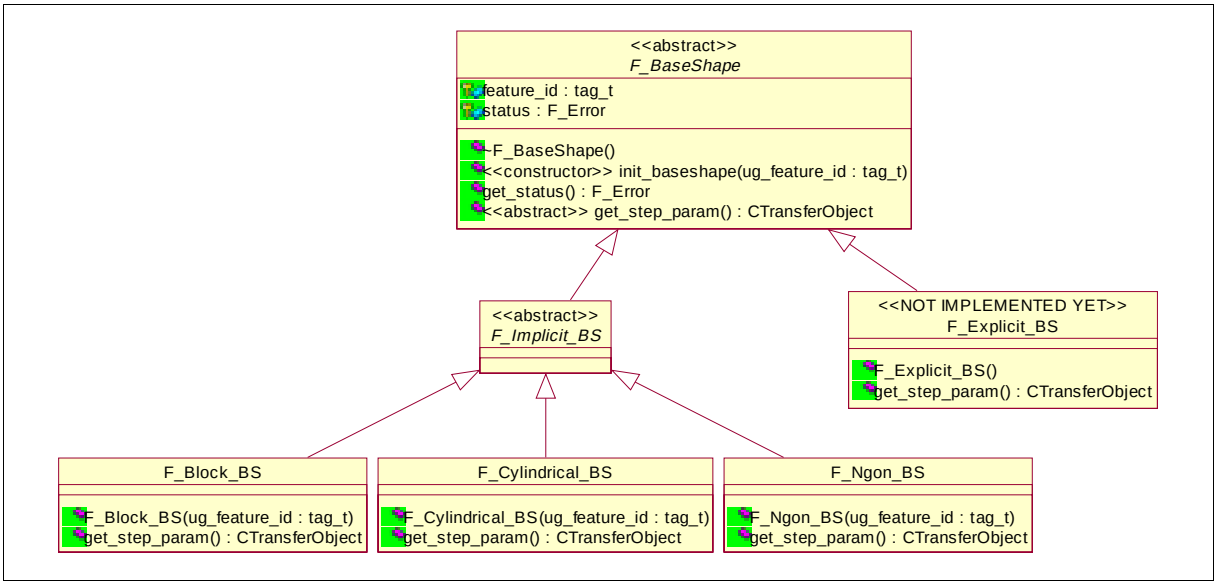


fig. 4-3 : static model for baseshapes

4.1.3 Process control

The classes representing features, faces and baseshapes are not sufficient for a software. They are just providing functionalities needed for a specific feature, a face or a baseshape. Now a special structure for the control of the program is necessary to react on use-cases and carry out the resulting processes.
To carry out the processes, objects of the previously described classes are instanciated whenever their functionality is needed. Thereby the appropriate methods can be called. But these objects are generally short-lived. After a special functionality for an specific object was used (e.g. create feature, get STEP parameters, etc.), they are deleted again.

Core of the control structure is the class `F_Control`. It contains a specific method for each use-case. In these methods, the processes for the use cases are performed. An instance of this class (`festeval_control`) is known to the callback-functions of the menu buttons. The callback-functions are calling the appropriate methods in `festeval_control`, and thereby the processes are initialised.
The class structure of the control system and its connection to the UG menus, the AP224_IM library for the STEP database and the classes for features, faces and baseshapes are shown in fig.4-4.

The file `F_Tools.h` provides some global functions which are helpful general tools. This includes the determination of the needed object for a specific feature, the determination of the UG features associated with a baseshape, a dialog for user decisions and some more tools.
In the file `Festeval_Types.h` several enumeration types are defined for identification of types of features and baseshapes and for status notifications.

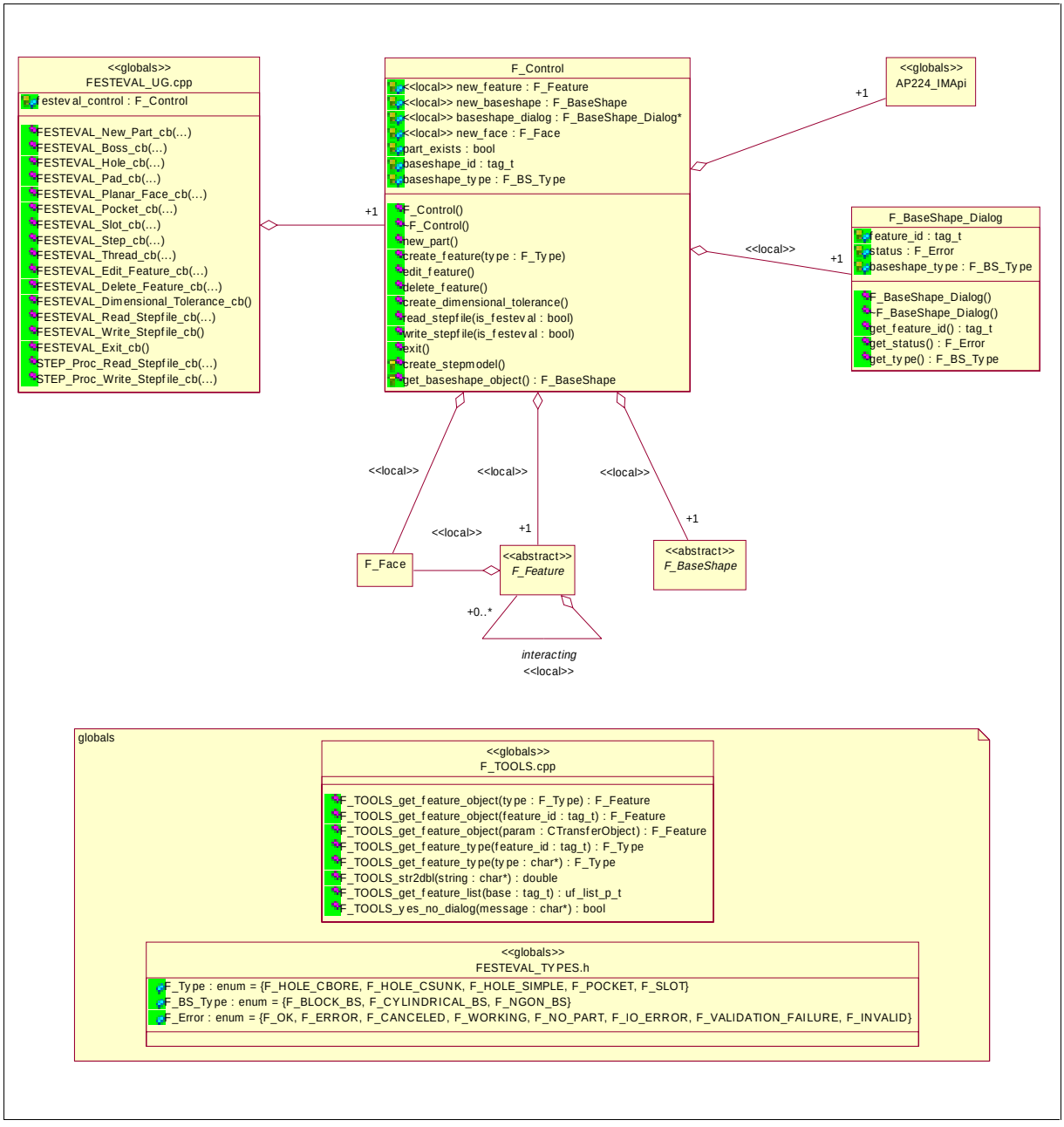


fig. 4-4 : static model for control

4.1.4 Complete static model

The complete static model with all classes, relations between the classes and files with global function are shown in fig. 4-5. The detailed definition of the classes can be seen in the previous diagrams. This diagram just provides an overall view of the complete system.

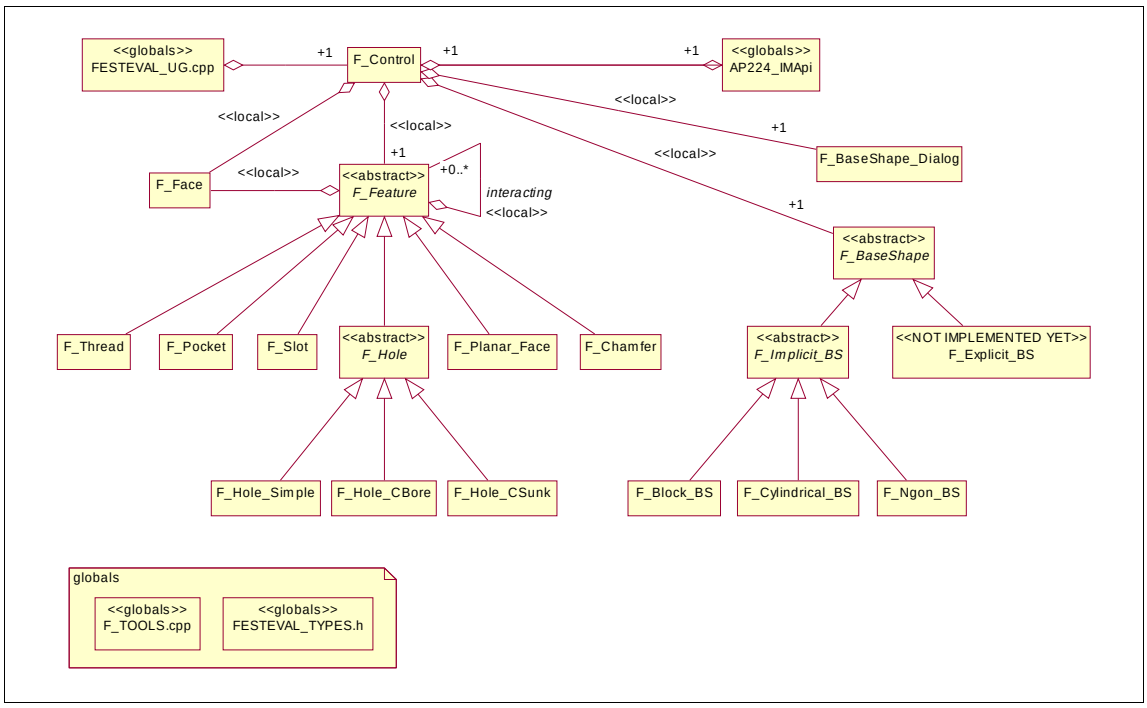


fig. 4-5 : complete static model

4.2 Dynamic model

While classes are representing a static structure, thus they do not vary during run-time, objects are representing a dynamic structure.

> Objects are instances of classes and are created, deleted and altered during run-time. They are the only existing constructs during run-time (everything is an object). At each moment they represent a certain state.
> The behaviour of objects during run-time is represented in the dynamic model.

The dynamic model visualizes, at what time which objects are created, deleted or used by other objects. A chain of such interactions always has an initialisation. In dynamic models these initialisations are usually use-cases. The dynamic model is used to check if the static model provides enough functionality to satisfy all requirements.

The dynamic behaviour is represented in UML interaction diagrams. An interaction diagram can be created for each use-case. There are two different kinds of interaction diagrams, which are representing the same circumstances, but in a different arrangement :

a) sequence diagrams : the constructs are arranged in chronological order.

b) collaboration diagram : the chronological order is represented by indices, the arrangement can follow other rules.

Usually collaboration diagrams seldom provide a better overview. In this thesis only sequence diagrams are used. In the following sections sequence diagrams are shown for the use-cases

- *new part*
- *create feature*
- *write STEP file*

The concerned objects are represented by rectangles at the top of the diagram. Each object has a life-line going from the object to the bottom of the diagram. During the livelihood of the object, this line is changing into a narrow rectangular. The livelihood starts with the instanciation (the call of the constructor) and ends with the deletion (the call of the destructor).
Communication between object is always done by methods of one object beeing called by other objects. This communication is represented by an arrow starting at the live-line of the object who calls a method (initialises a communication). The arrow ends at the life-line of the object whose method is called. The arrow is labeled with the name of the method and the parameters. An object can also call methods of itself. This is represented by an arrow inside the live-line.
Communication is only possible during the livelihood of an object.

To consider the procedure oriented code which was used in the software, global functions or their files are also represented as objects, and calls of global functions are represented in the same way as calls of methods. This is the case for the callback-functions in all diagrams and the functions of the AP224_IMApi in fig. 4-8.

4.2.1 New part

The sequence diagram for this use-case can be seen in fig. 4-6. This use-case is initialised by a call of the method `new_part()` of the object `festeval_control` by the callback-function `FESTEVAL_New_Part_cb()`. The control object is then instanciating the object `baseshape_dialog`, which is used to determine the baseshape information from the user. After the required information is determined by calls of `get_feature_id()` and `get_type()`, the object `baseshape_dialog` is deleted again. Then for each feature connected to the choosen baseshape an object `festeval_feature` of the appropriate sub-class of *`F_Feature`* is instanciated. The constructor `F_Feature(feature_id)` is used, because the feature already exists in the UG database (see also case b) in chapter 4.1.1). The instanciated object is used to validate the feature by a call of the method `validate()` and is then deleted again. Thereby all features are validated to assure that no invalid features are used within the *FESTEVAL* environment.

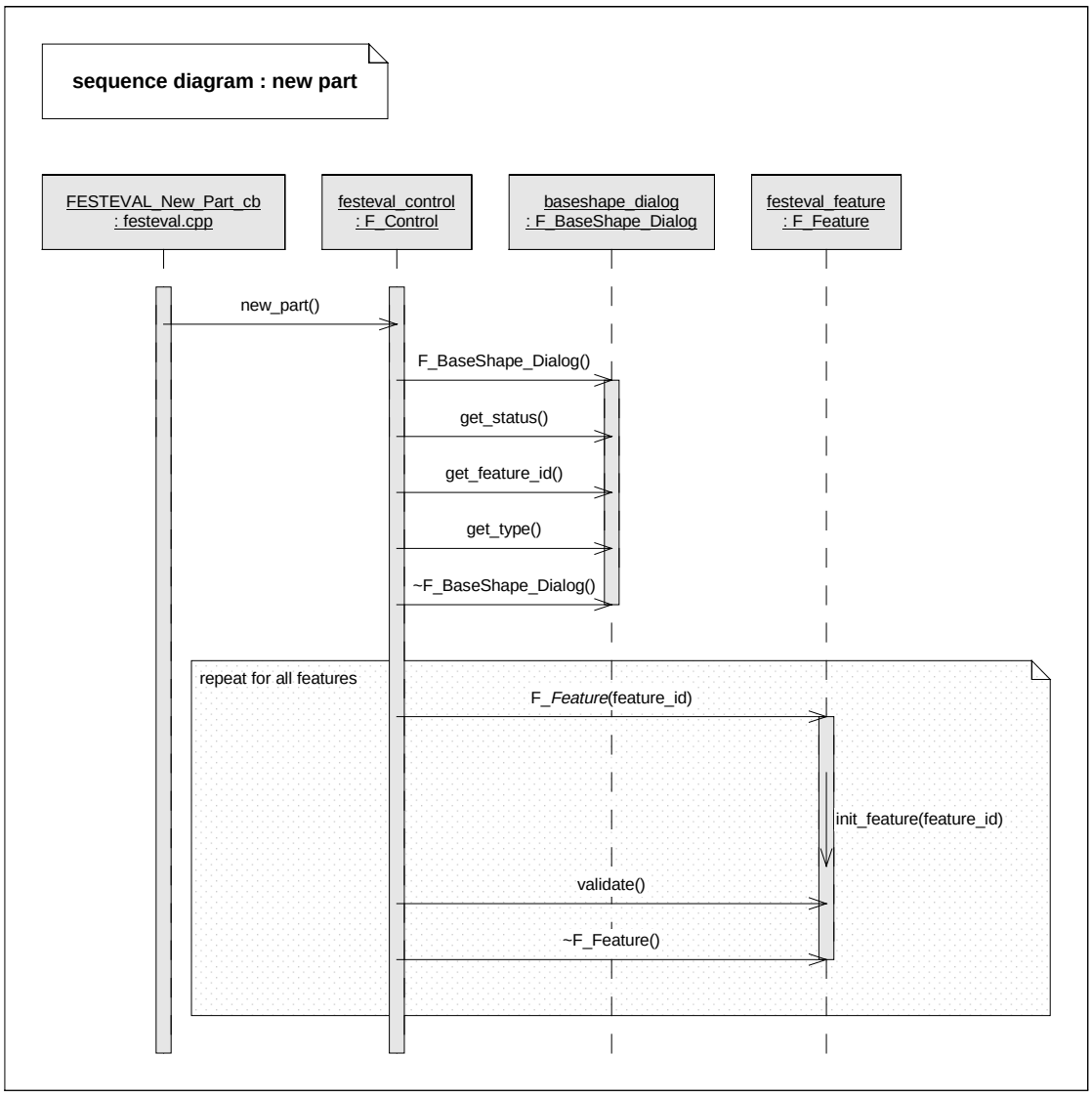


fig. 4-6 : sequence diagram new part

4.2.2 Create feature

The sequence diagram for this use-case can be seen in fig. 4-7. The initialisation is done by a call of the method `create_feature(type)` of the object `festeval_control` by the callback-function `FESTEVAL_Create_Feature_cb()`. Tha parameter `type` specifies the type of feature that is to be created. The method `create_feature(type)` then instanciates an object of the appropriate sub-class of *`F_Feature`*. Here, the constructor `F_Feature()` is used, because the feature does not exist at this time (see also case a) in chapter 4.1.1). The method `init_feature()` of the object `festeval_feature` is called automatically by the constructor. It calls the method `create_ug_feature()`, which handels the user dialog for the parameters and the creation of the feature in the UG database. Finally, a validation is done by a call of the method `validate()`. At the end of this use-case, the object `festeval_feature` is deleted again.

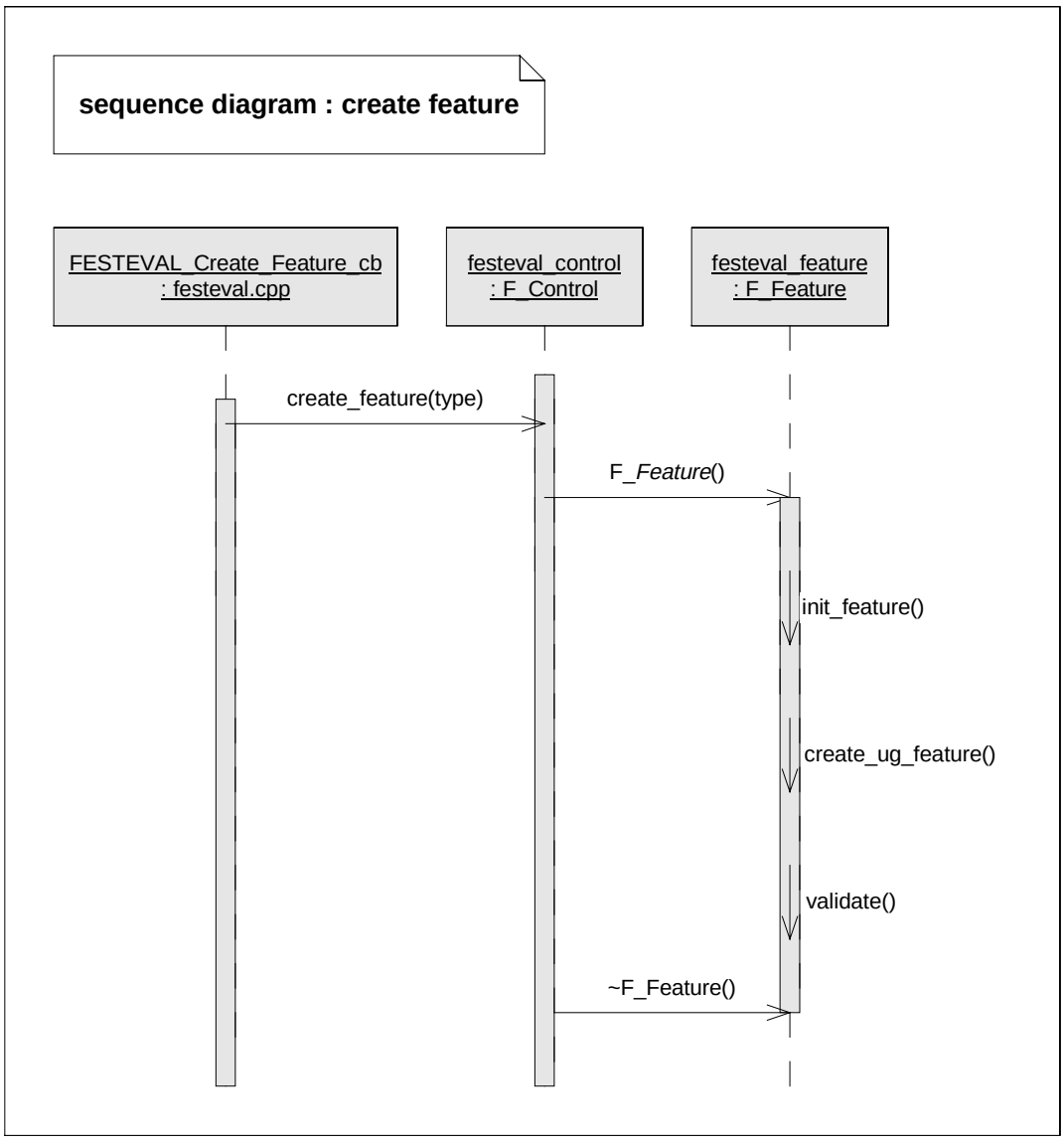


fig. 4-7 : sequence diagram create feature

4.2.3 Write STEP file

Besides the actual creation of the physical file, this use-case also includes the creation of the STEP database. All necessary information has to be determined, either from the UG datamodel or by appropriate methods, and to be stored in the STEP datamodel.
The sequence diagram can be seen in fig. 4-8. The use-case is initialised by a call of the method `write_stepfile()` of the object `festeval_control` by the callback-function `FESTEVAL_Write_Stepfile_cb()`. The STEP database is cleared by a call of the function `IMClearModel()`. Then the creation of the STEP database is initialised by a call of the method `create_step_database()` of the object `festeval_control`. This first instanciates an object of the appropriate sub-class of *`F_Baseshape`*, which is used to determine the parameters of the baseshape with its method `get_step_param()` and is then deleted again. The baseshape is created in the STEP database by a call of the function `IMCreateBaseshape(baseshape_param)`, the previously determined paramters are given to that function.
For each feature of the part the object `festeval_feature` of the appropriate sub-class of *`F_Feature`* is instanciated using the constructor `F_Feature(feature_id)` (see also case b) in chapter 4.1.1). For each face of the actual feature, an object `new_face` of the class `F_Face` is instanciated. This object is used to determine the parameters of the face (`get_step_param()`), which are used to create the face in the STEP database (`IMCreateFace(face_param)`). Then the parameters and constraints of the feature itself are determined (`get_step_param()` and `get_constraints_param()` of the object `festeval_feature`) to create the feature in the STEP database (`IMCreateFeature(feature_param)` and `IMAddConstraints(constraints_param)`).
After all this information is created in the STEP database by the explained means, the physical file is created by a call of the function `IMWriteFile(filename)` and the STEP database is deleted again by a call of the function `IMDeleteInstances()`.

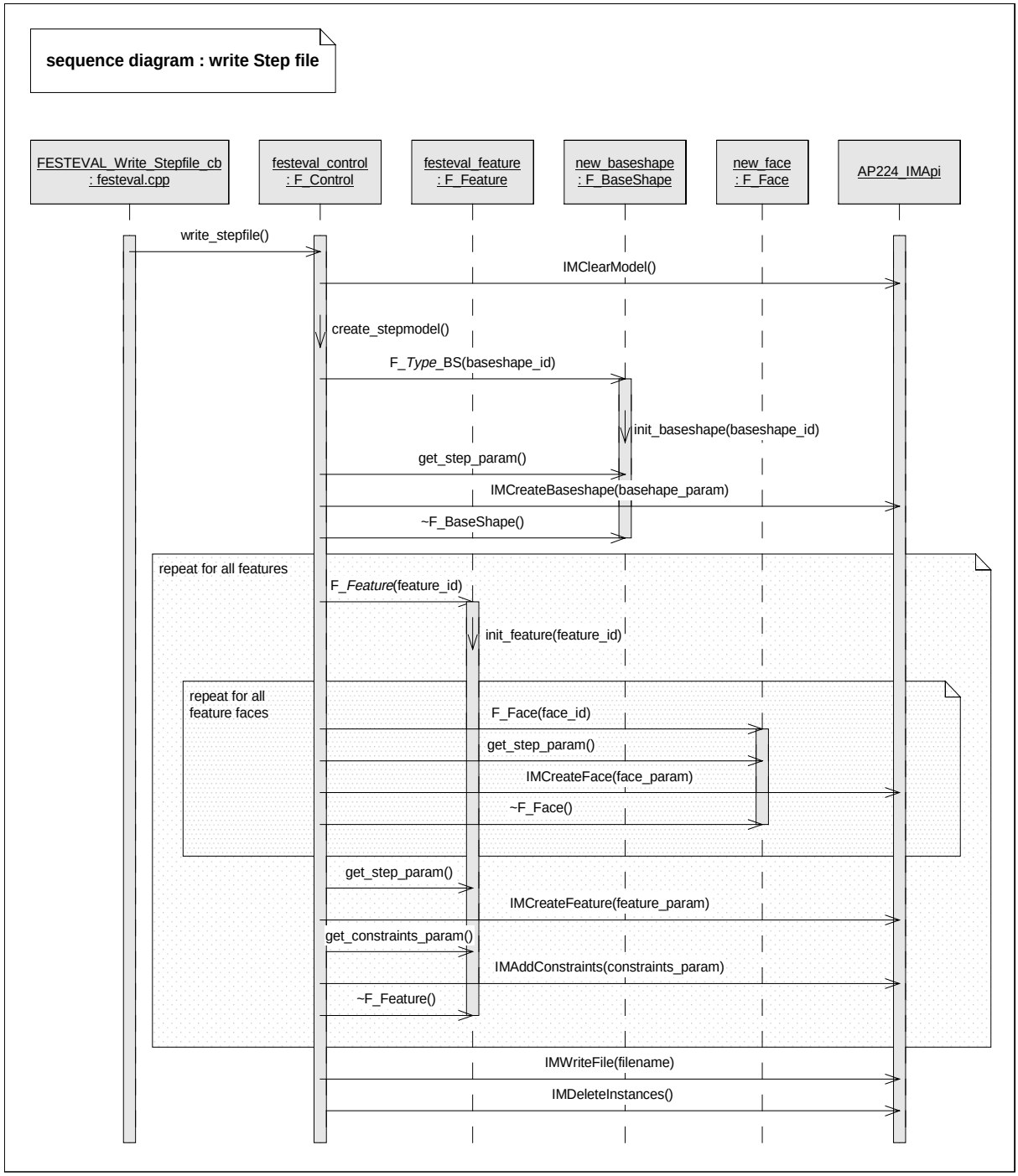


fig. 4-8 : sequence diagram write STEP file

5. Implementation

In the implementation stage, the object-oriented model is transferred into an object-oriented programming language.

It can again be distinguished between the static and the dynamic structure.

5.1 Static structure

The static structure can be derived from the information contained in the static model. This structure consists of :

- classes
- attributes
- definition of methods (name / type / parameters)

Modern UML tools can automatically create sourcecode from the class diagrams in any object-oriented programming language. But even if the coding is done manually, the implementation of the static structure is trivial, because the code can directly be derived from the class diagrams.

5.2 Dynamic structure

The dynamic structure of the programm code is representing the actual processes the software can perform, thus the bodies of the methods. Even if a (small) part of the information about the dynamic structure is contained in the dynamic model, an automatic implementation makes only very limited sense. The reason is that the dynamic model contains only the interactions between the objects, but not the processes that are performed by the methods.
Normally the bodies of the methods are coded manually using ASCII editors, which can be specific to the used programming language. An example is the editor integrated in the environment of Visual C++.

Methods (and functions) in structured programming languages are assembled by three types of control structures :

- sequence
- bifurcation
- loop

For the conception of methods, structograms (Nassi-Schneidermann) are used. This type of diagram provides a graphical notation for the different kinds of control structures. Complex methods (methods with at least one selection or repetition) should be planned carefully before implementation. In the following sections structograms for the five methods

- `F_Control::new_part()` (controls use-case *new part*)
- `F_Control::create_feature()` (controls use-case *create feature*)
- `F_Feature::init_feature()` (creates a new feature in Unigraphics)
- `F_Control::write_stepfile(bool is_festeval)` (controls use-case *write STEP file*)
- `F_Control::create_step_database()` (creates the STEP database)

are shown. They are arranged by the use-cases for which they are used.

5.2.1 New part

`F_Control::new_part()`

See fig. 5-1 for the structogram of the method `F_Control::new_part()`. This method controls the operations that have to be done if the user wants to start a new session in the *FESTEVAL* environment. A running session is indicated by the boolean class attribute `part_exists` of `F_Control` beeing true. The use-case *new part* can only be performed if no session is started yet. If this is the case, a baseshape is determined with the object `baseshape_dialog` of the class `F_Baseshape_Dialog`. Only if the status returned by this object has the value `F_OK`, the operation continues. For other states errors messages are displayed.
If a valid baseshape was choosen by the user, its identifier and a list with the identifiers of all connected features are determined. For each feature an object of the appropriate sub-class of *`F_Feature`* is instanciated inside a loop. This object is used to validate the feature and is then deleted again. The identifiers of features of unknown type and of invalid features are put in the list `del_list`. If unknown or invalid features were found, the user has to decide via a dialog if he wants to delete these features. Only if he decides to do so, a session is started by setting the attribute `part_exists` to true.

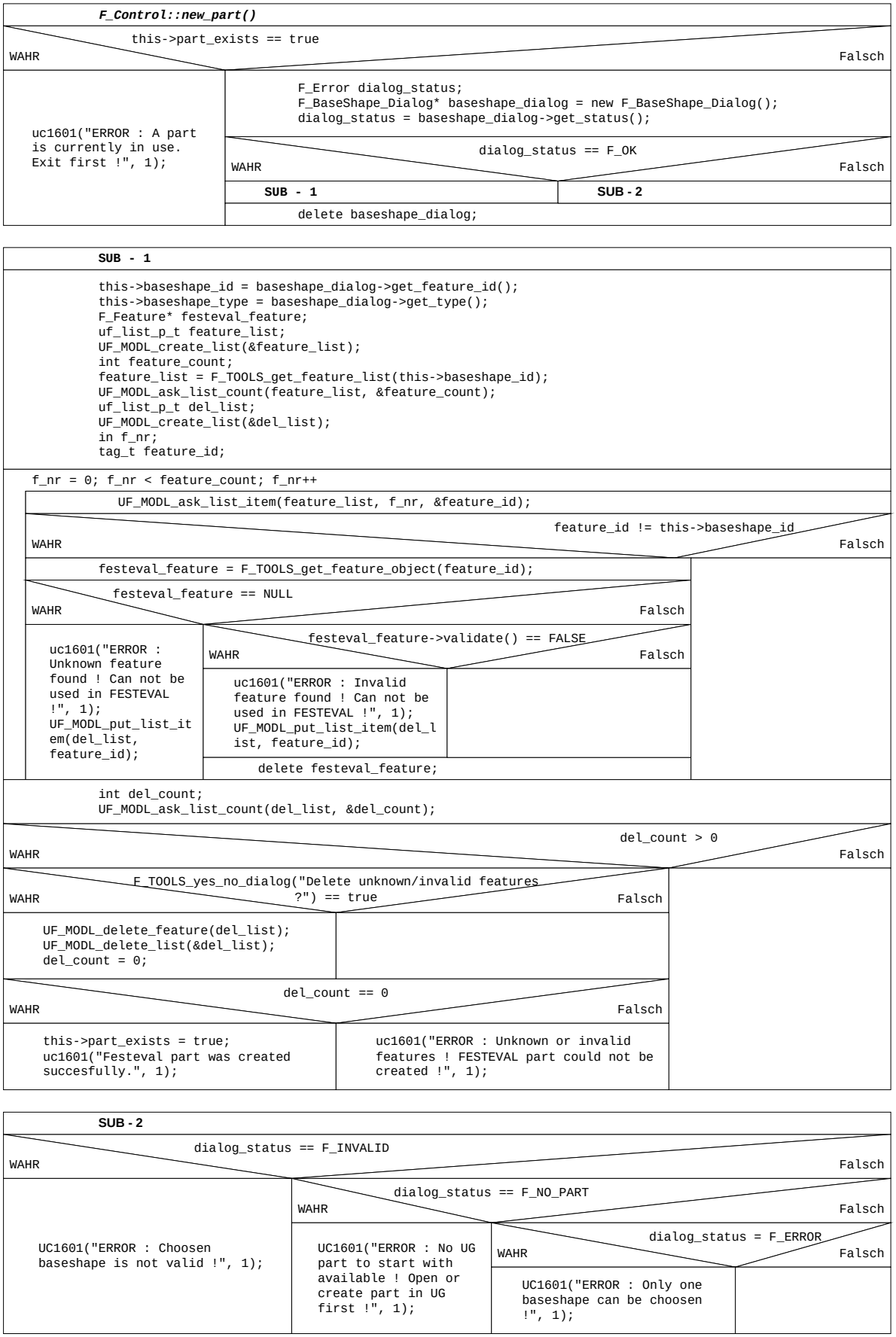


fig. 5-1 : structograms new part

5.2.2 Create feature

The method `create_feature(F_Type type)` of the class `F_Control` controls operations that have to be done if the user wants to create a new feature. The parameter `type` specifies the type of the new feature. The main operations are done inside the method `init_feature()` of the class *`F_Feature`*, so this method is also described here. The structograms can be seen in fig. 5-2.

`F_Control::create_feature(F_Type type)`

This functionality can only be used when a session in the *FESTEVAL* environment was started. If this is the case, the object `festeval_feature` is instanciated with the appropriate sub-class of *`F_Feature`*. The sub-class appropriate to the type of feature is determined by a call of the method `F_TOOLS_get_feature_object(F_Type type)`. The constructor of the new object calls the `method init_feature()` of the class *`F_Feature`*.

`F_Feature::init_feature()`

This method controls the operations for creating a new feature of the type according to the class which was used to instanciate the object. After some settings are done, the method `create_ug_feature()` is called. This method is implemented not in the abtract class *`F_Feature`* but in its concrete sub-classes. Thereby it has a different implementation for each type of feature. If, for example, the concrete sub-class `F_Pocket` was used to instantiate the object `festeval_feature`, the implemenetation of `create_ug_feature()` in the class `F_Pocket` will be called. This method creates an user dialog for the feature specific parameters and creates the feature in the UG database. It is similar to the original Unigraphics functionality for creating a feature.
After the feature was created, it is validated by a call of the method `validate()`. This method is again different for each type of feature, and is therefore implemented in the concrete sub-classes of *`F_Feature`*. If the feature is valid, all interacting features are determined by the call of `get_interactions()`. A call of the method `validate_interacting_features()` then validates these features. If it returns `NULL`, all interacting features are valid.
If either the new feature or one if its interacting features are invalid, the new feature can not be used in the *FESTEVAL* environment. In this case an error message is displayed and the new feature is deleted again.

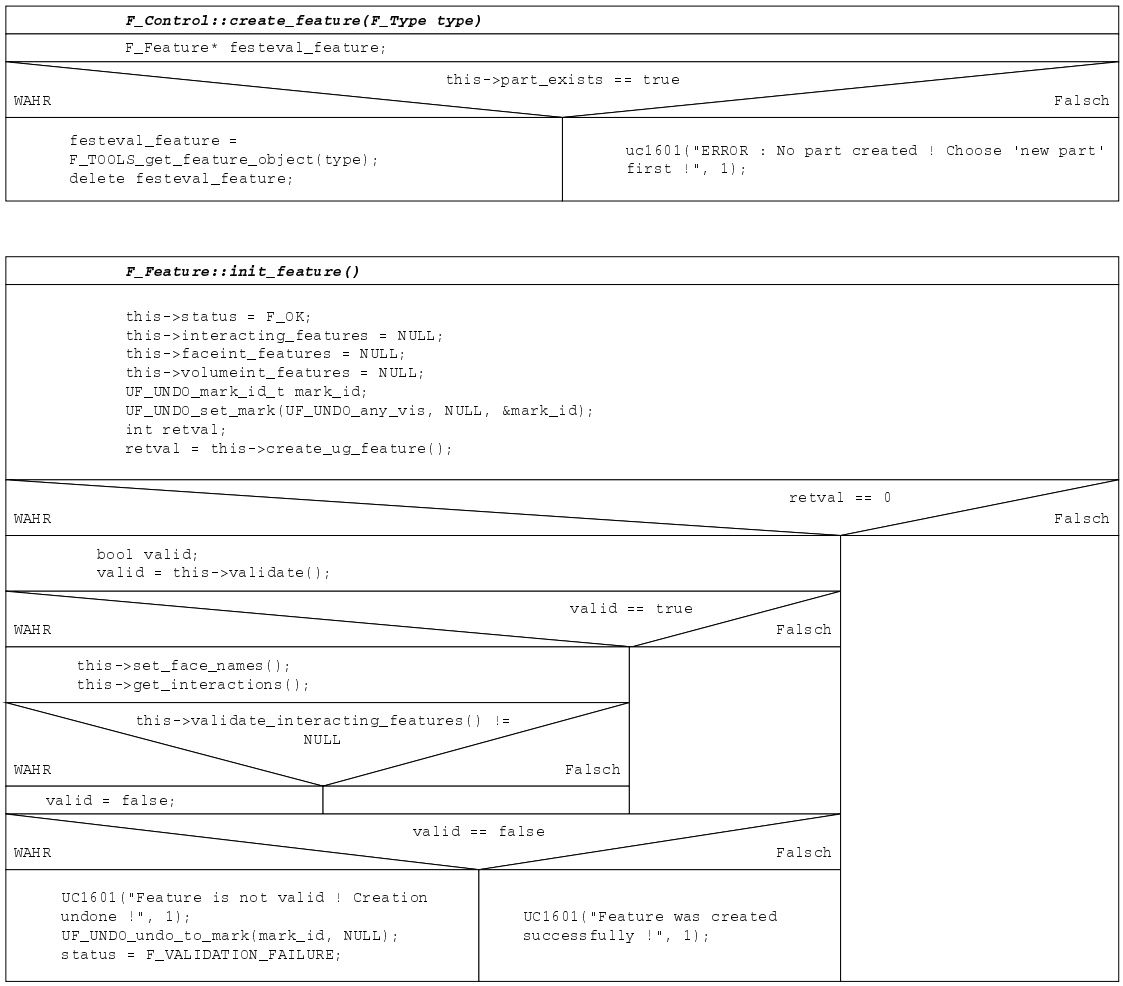


fig. 5-2 : structograms create feature

5.2.3 Write STEP file

The operations performed to create a physical STEP file containing the product data are controlled in the method `F_Control::write_stepfile(bool is_festeval)`. The main point of this functionality is to transfer the product data from the UG database to a STEP conform database. This is done in the method `F_Control::create_step_database()`. The structograms of these methods can be seen in fig. 5-3 and fig. 5-4.

`F_Control::write_stepfile(bool is_festeval)`

The parameter `is_festeval` indicates whether the method was called within the *FESTEVAL* environment. In this case, the part choosen in the use-case *new part* is used, otherwise the user has to choose a part from the Unigraphics database, like described in chapter 5.2.1. This is performed in the part-diagram **SUB-1**.

The STEP file can only be created if a part exists, e.g. a FESTEVAL session must be running or a part must have been choosen if the method was called outside of FESTEVAL.
If a part exists, the user first has to choose a location and has to enter a filename for the STEP file. If this was successful, the STEP database is created by a call of the method `create_step_database()`, the STEP file is written (`IMWriteFile(filename)`) and the STEP database is deleted again (`DeleteInstances()`).

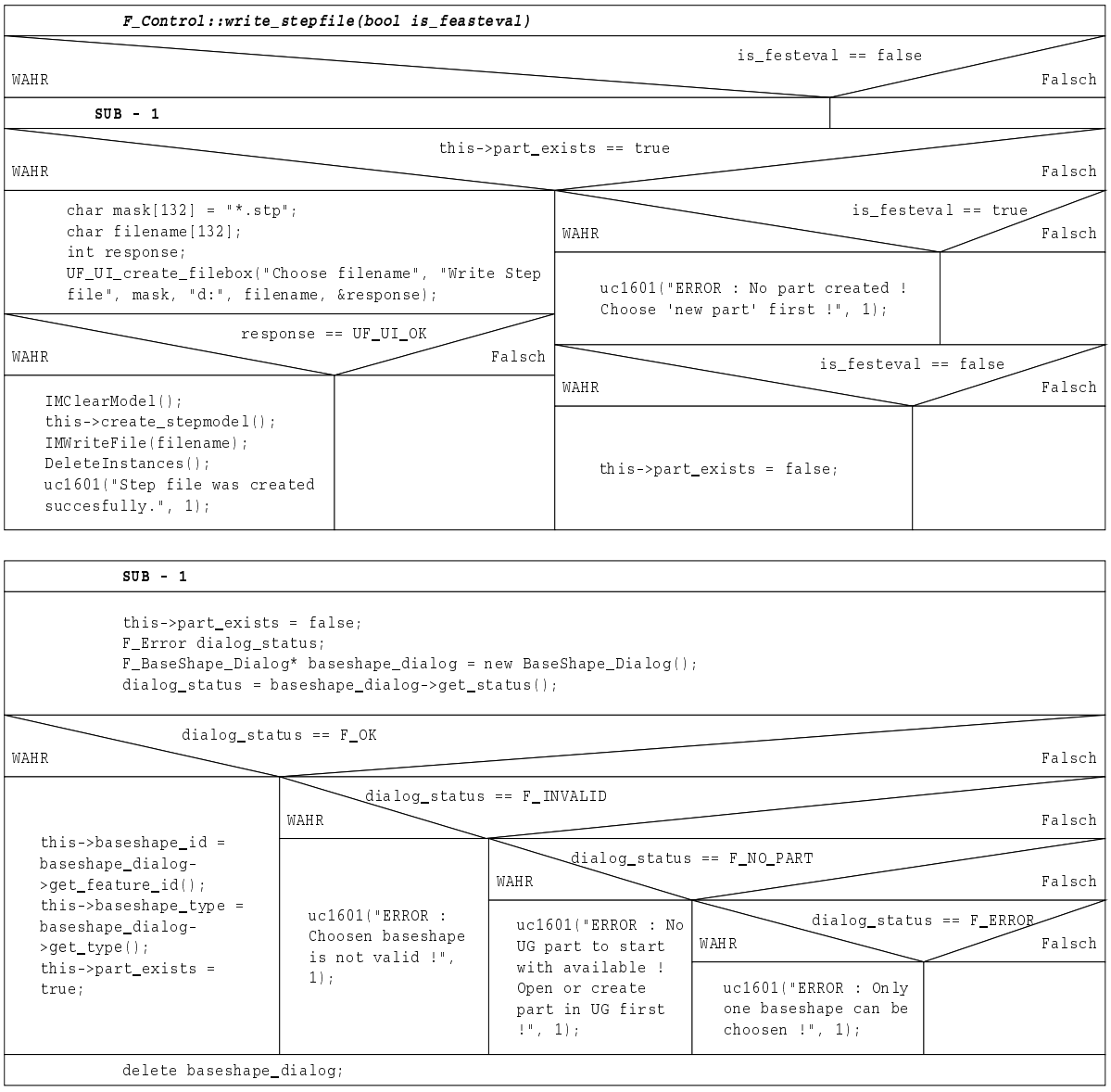


fig. 5-3 : structograms write STEP file

F_Control::create_step_database()

In this method the information needed for the STEP database is determined either from the UG database or from appropriate methods, then it is used to create instances in the STEP database.
First, the baseshape is handeled. The object `new_baseshape` is instanciated with the appropriate sub-class of *F_BaseShape*, which is determined with the method `get_baseshape_object()`. The object `new_baseshape` is used to determine the parameters of the baseshape by a call of the its method `get_step_param()`. The determined parameters are used to create the instance in the STEP database (`IMCreateBaseShape(baseshape_param)`) and the objects `new_baseshape` and `baseshape_param` are deleted again.
Then a list with all features connected to the baseshape is determined using the function `F_TOOLS_get_feature_list(this->baseshape_id)`. In a loop for each feature an object of the appropriate sub-class of *F_Feature* is instanciated. An error message occurs, if the type of feature is unknown.
For most types of features, faces have to be created in the STEP database. This is done in the part diagram **SUB-1**. After a list with the faces of the feature was created, the object `new_face` of the class `F_Face` is created for each face. This object is used to determine the parameters of the face by a call of its method `get_step_param()`. These parameters are used to create the face by a call of the function `IMCreateFace(face_param)` and the index of the instance in the STEP database is added to the entry in the UG database (`add_ug_attr_int(...)`). This is necessary to create a link from the feature to the face later. The objects `new_face` and `face_param` are deleted again.
After the faces were created, the parameters of the feature itself are determined by a call of the method `get_step_param()` of the object `festeval_feature`. The instance in the STEP database is created by a call of the function `IMCreateFeature(feature_param)`, its index is added to the entry in the UG database (`add_ug_attr_int(...)`) and the objects are deleted again.
After all features and their faces are created in the STEP database, another loop is necessary to store the information about the interactions in the STEP database. The extra loop must be used to make sure that features that are referenced are already existent in the STEP database. Again, an object `festeval_feature` is created for each feature. To get the information about the constraints, the method `get_constraints_param()` in this object is called. The determined parameters are used to add the constraints to the instance in the STEP database by a call of the function `IMAddConstraints(step_index, feature_param)`.

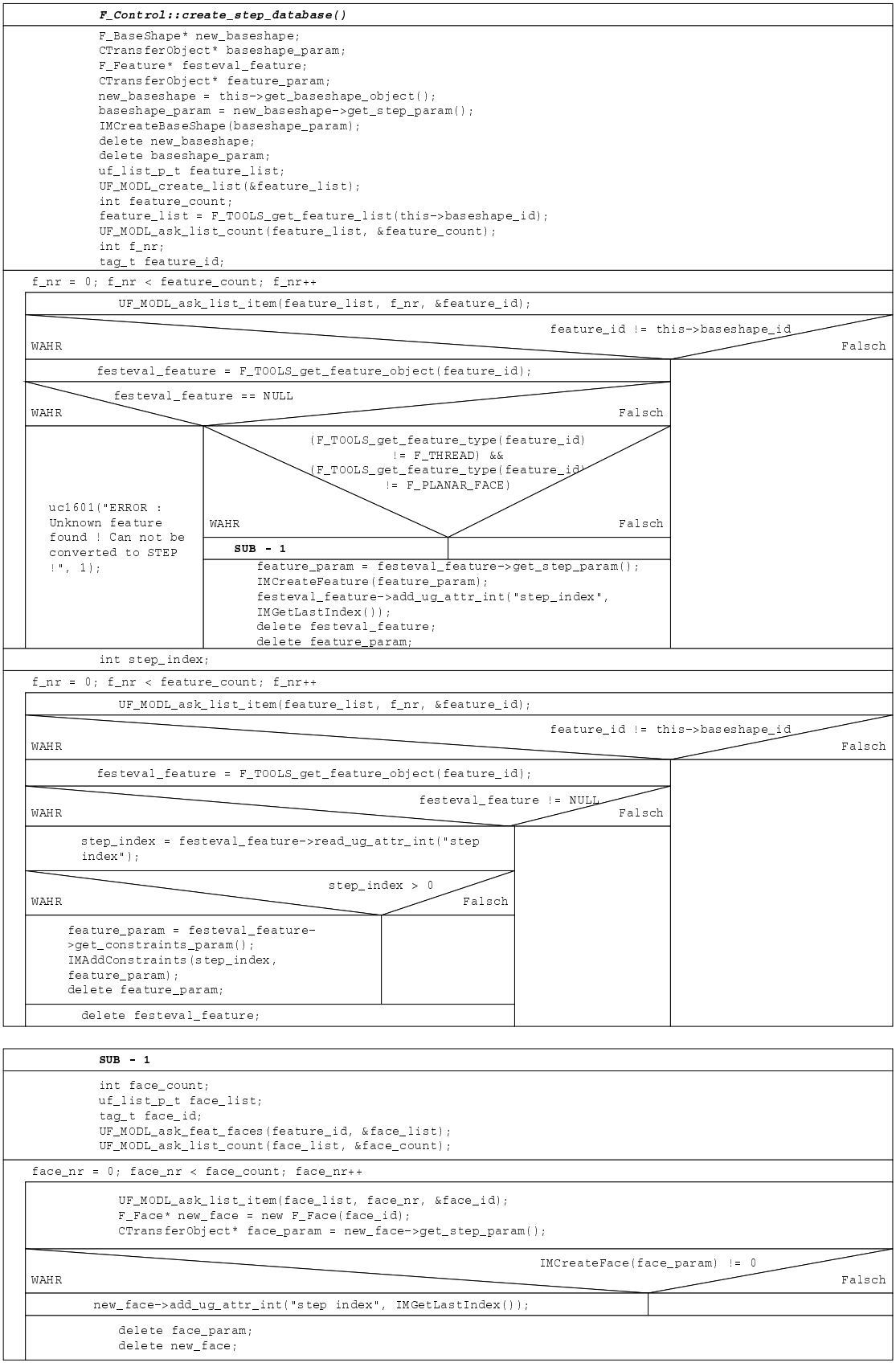


fig. 5-4 : structograms create STEP database

5.3 Menus

The GUI (graphical user interface) is the interface between the user and the software. In this thesis, the GUI consists of the GUI of Unigraphics and the new menus of the FESTEVAL environment.

The new menus are created with the *UG user interface styler*, which can be launched from Unigraphics. The buttons of the main menu of the FESTEVAL environment can be seen in fig. 5-5. Each use-case has an appropriate button. The use-cases described in this documentation are launched by the buttons New Part, the buttons Create Feature and the button Write Stepfile. Further use-cases handeled in this project are edit and delete feature, handling of technological attributes, reading STEP files and exit the FESTEVAL environment.

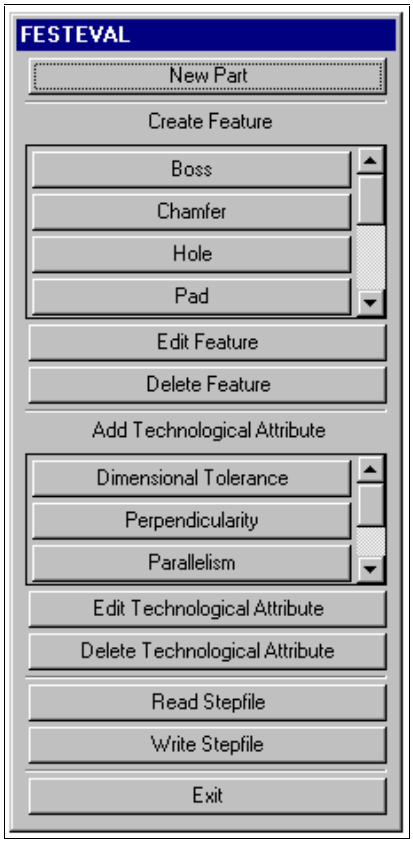


fig. 5-5 : main menu of the FESTEVAL environment

Each button of the menus is connected to a callback-function, which is automatically called when the user presses the appropriate button. The sourcecode files with these callback-functions are created automatically by the UG user interface styler.
The connection between this procedure-oriented structure and the object-oriented structure of the FESTEVAL software is materialised by an object of the class `F_Control`, which is known to the callback-functions. Each callback-function calles an appropriate method in this object.

The flow of actions is visualized in fig. 5-6.

Example : The user presses the button Write Stepfile. Unigraphics automatically calls the appropriate callback-function `FESTEVAL_Write_Stepfile_cb(...)` *in the file* `festeval_ug.cpp`. *The callback-function calls the method* `write_stepfile(true)` *of the object* `festeval_control`. *This method initialises and controls all operations that are necessary for this use-case.*

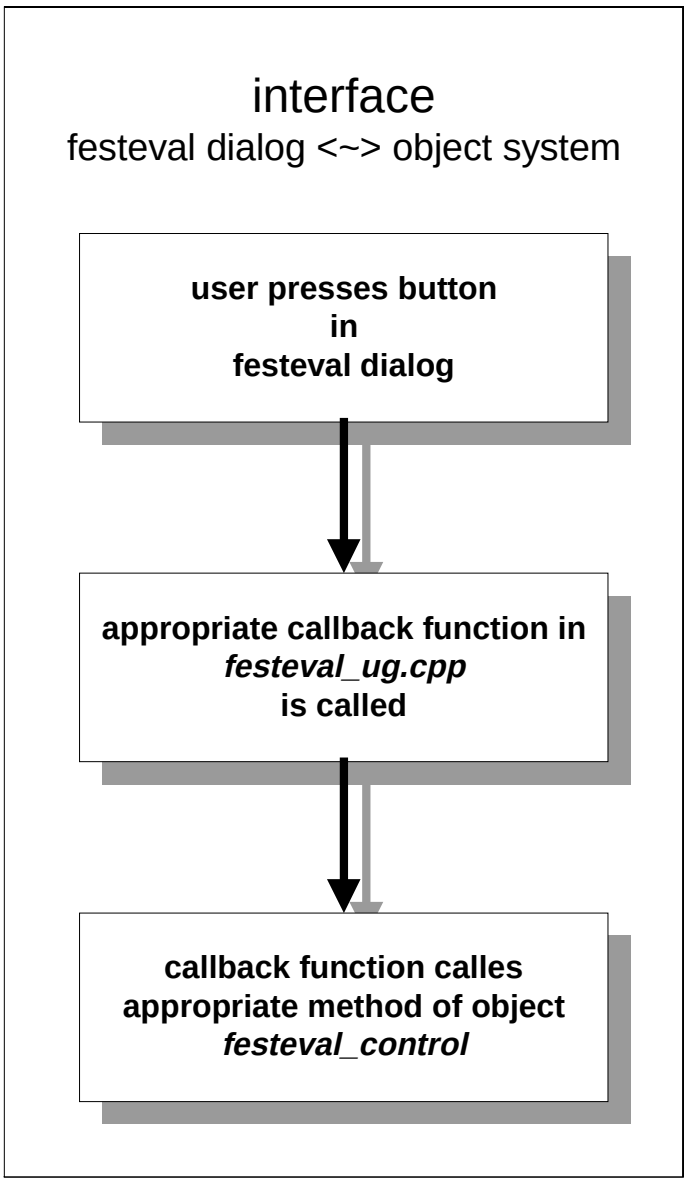


fig. 5-6 : connection menu <–> object system

6. Use

In the use stage, the developed software is used either as a prototype in a testing environment, or as a release version in commercial use. Results are information about : - bugs - new requirements

In this project, extensive tests with the developed software were accomplished. In further transverses through the development cycle the results of the tests were considered, and thereby the structures were optimized and the functionality extended until the requirements were fulfilled. Possible future requirements, which were not or only partly considered in this thesis, are described in the prospect.

In the following, screenshots of Unigraphics with the FESTEVAL environment where different functionalities are in progress can be seen. In all screenshots, a part is opened in the UG database and is displayed in the diply window of Unigraphics. This part consists of a block and a number of features.

new part

In the screenshot of fig. 6-1, the FESTEVAL function *new part* was choosen by the user. A dialog box can be seen, which allows the user to choose one of the features of the part opened in the UG database. The user is required to choose one of the features as a baseshape, which is required for a FESTEVAL part. Valid types of baseshapes are

- *block*
- *cylinder*
- *prism*

For the part used in this example, the only feature which would be a valid baseshape for FESTEVAL is BLOCK(0).

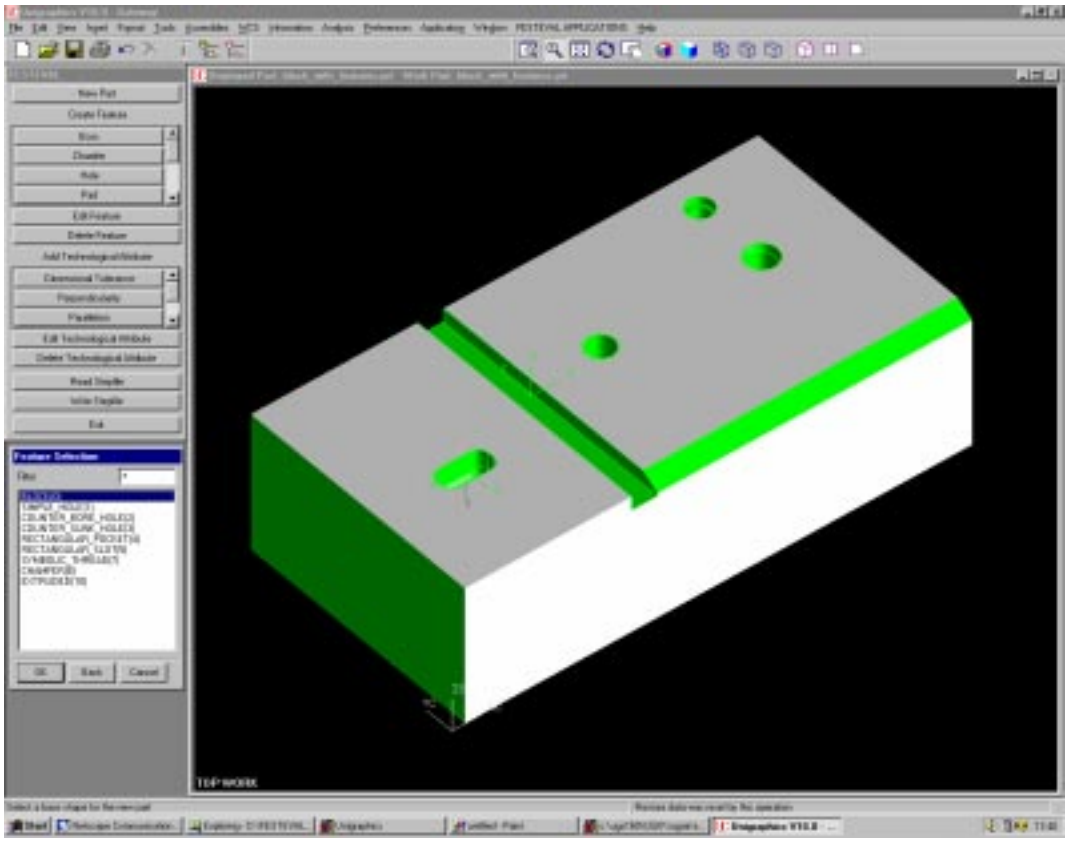


fig. 6-1 : screenshot new part

unknown feature (new part)

Every feature connected to the choosen baseshape has to be of a type known to FESTEVAL. At present, the following feature types are implemented :

- *thread (symbolic)*
- *pocket*
- *slot*
- *simple hole*
- *counterbore hole*
- *countersunk hole*
- *planar face*
- *chamfer*

If features of unknown types were found in the opened part, an error message is displayed, like shown in fig. 6-2. A similar case occurs if the feature does not satisfy the FESTEVAL validation. Such features can automatically be deleted to make the part usable in the FESTEVAL environment.

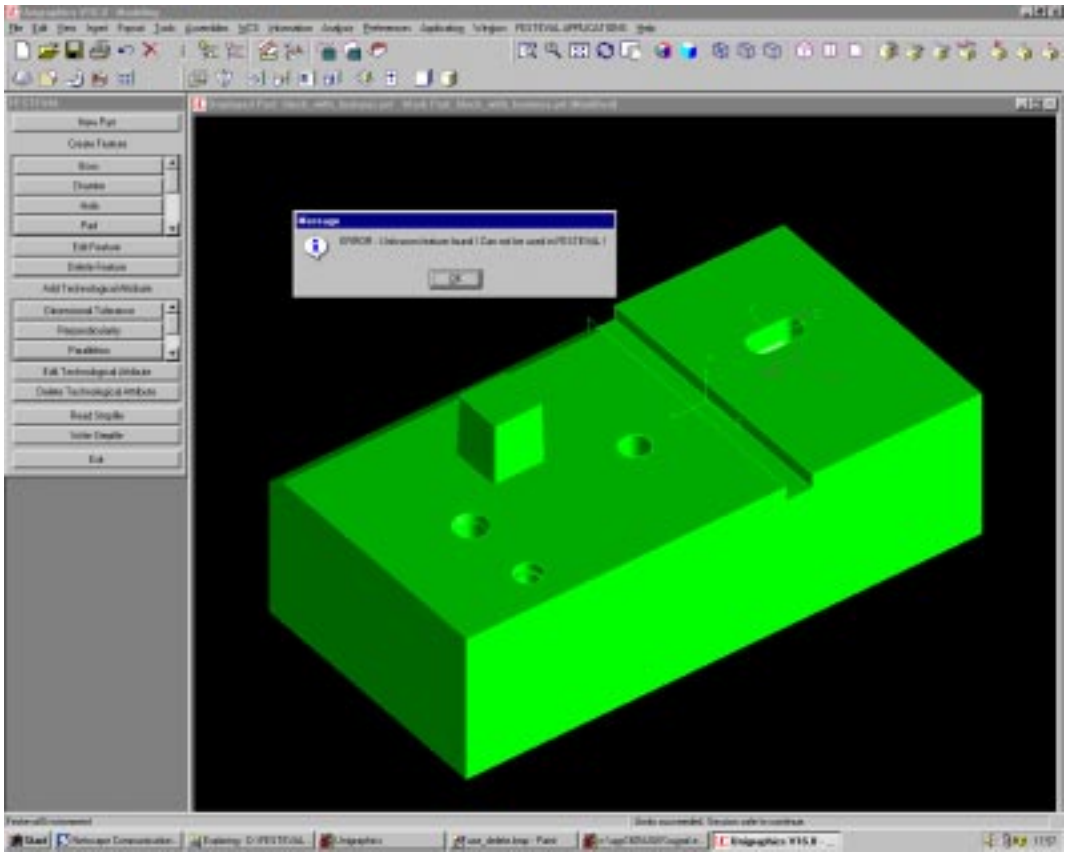


fig. 6-2 : screenshot unknown feature

new feature

In the screenshot shown in fig. 6-3, the functionality to create a new pocket was choosen by the user. The geometric parameters have to be entered in the dialog box shown. Another dialog box will occur to specify the placement for the new feature. After all parameters are entered, the new feature and all its interacting features will be validated according to the FESTEVAL specifications. If at least one of these features is invalid, the new feature is automatically deleted again and an appropriate error message is displayed.

delete feature

The screendhot in fig. 6-4 shows the dialog box that occurs after the functionality for deleting features was choosen. All the features of the current part are displayed, and the user can choose one or more features to be deleted. All child-features of the choosen features will automatically be deleted as well.

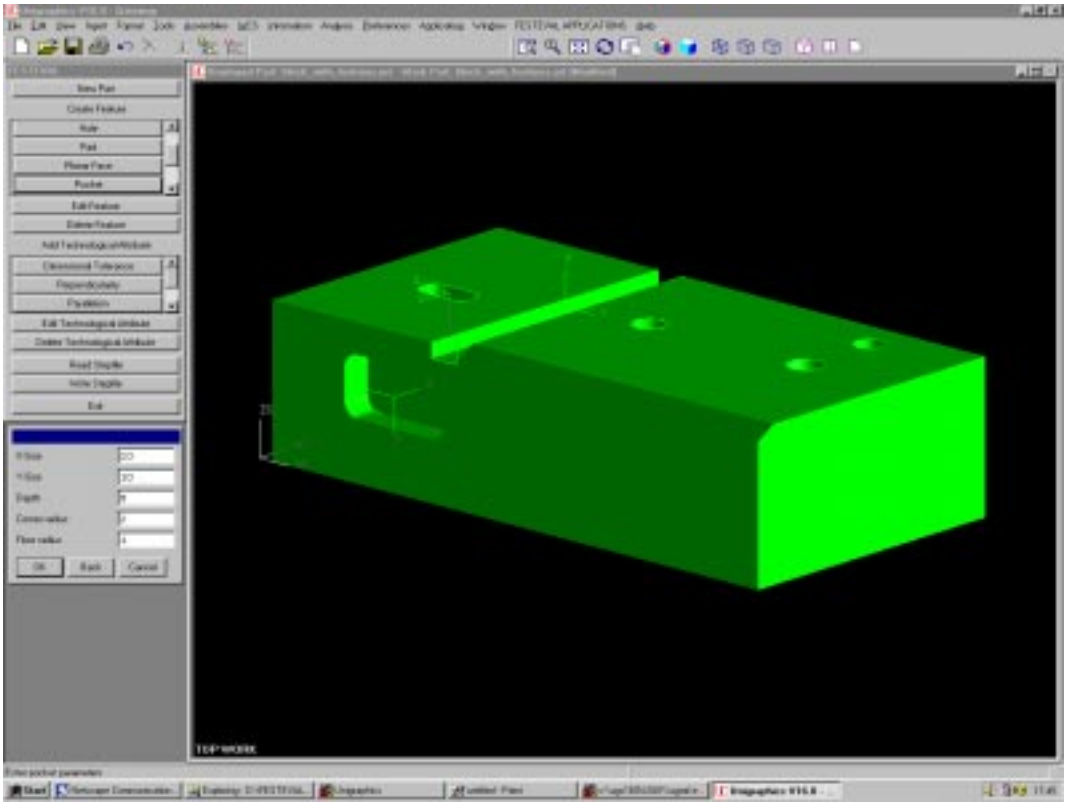


fig. 6-3 : screenshot create feature

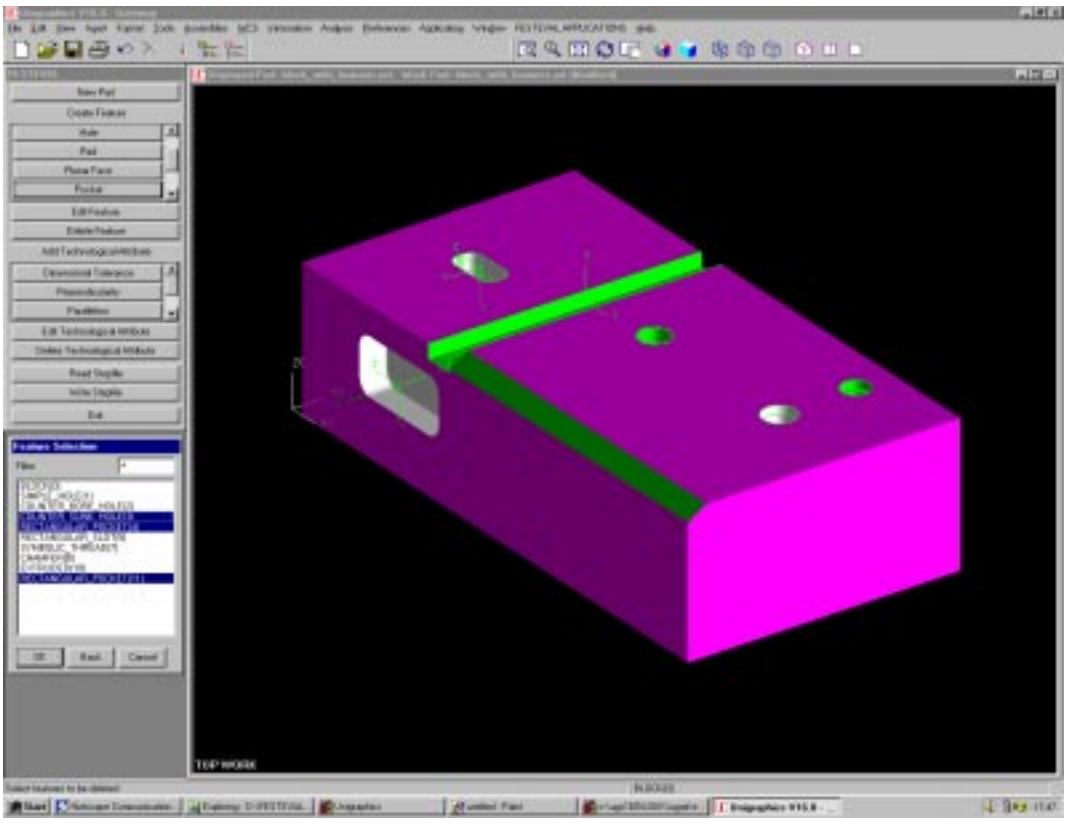


fig. 6-4 : screenshot delete feature

write STEP file

If the functonality for writing a STEP file is choosen, a dialog box for entering a location and a filename is displayed, like shown in fig. 6-5. After the user enters a valid location and name for the file, all necesary product information are transferred into a STEP database which is conform to the STEP datamodel used in FESTEVAL.
The STEP file is the connecting link between the different stages of the process chain. It can be read, interpreted and modified by the different CAx systems.

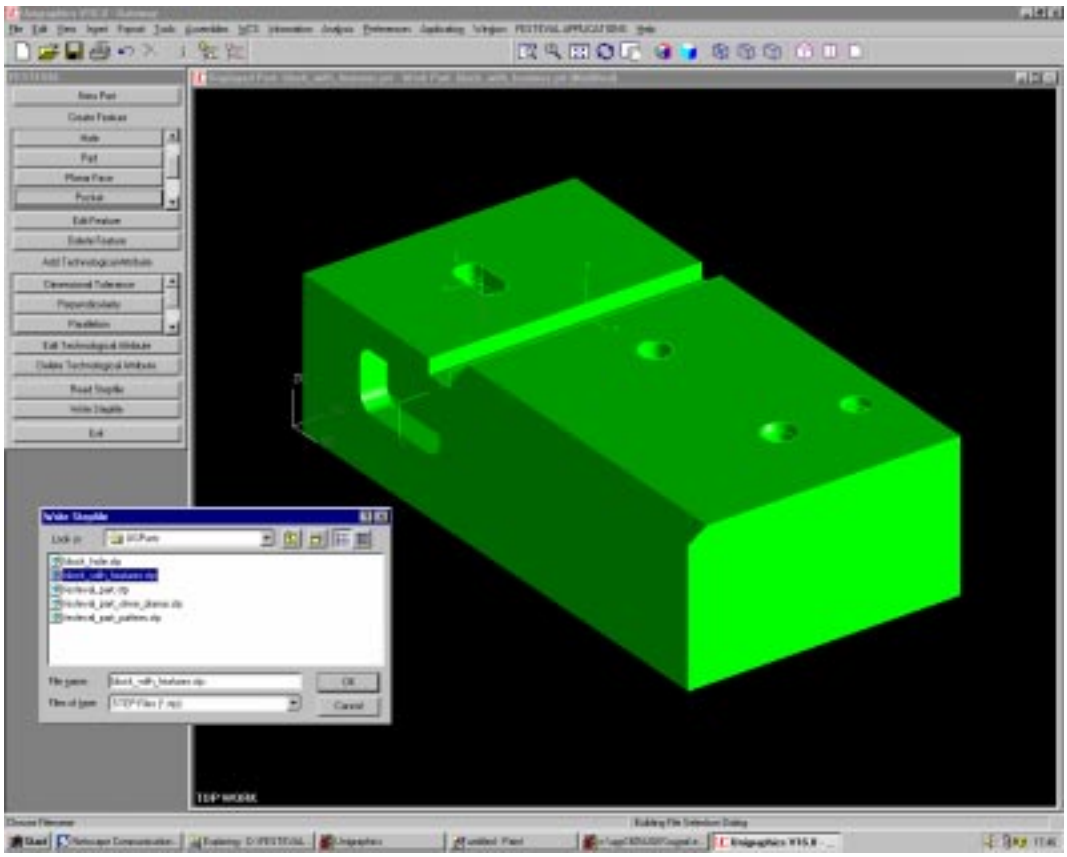


fig. 6-5 : screenshot write STEP file

7. Prospect

In this thesis a new design environment in Unigraphics was developed which integrates in the *FESTEVAL* process chain. The most vital functionality was implemented and the new environment is applicable for a series of parts.
The product data created in the new environment can be stored in a physical file which is conform to the STEP standard. This file can be read by the next station in the process chain.
The developed software is a basis and framework for future enhancements for *FESTEVAL* or succession projects.
Suggestions for enhancements and proposals for their solutions are given in this chapter.

The proposals for solutions enlisted here are just suggestions how to start dealing with the problem. Wether they really prove to be suitable can only be found out by the attempt. All stages of the software development should be considered during enhancemants.

7.1 Constraints

In the *FESTEVAL* datamodel a number of constraints can be assigned to each feature (see fig. 3-4). Constraints are geometrical or technological interdependencies and access information for machine tools. The environment developed in this thesis handles the two types of constraints `volume_interaction` and `face_interaction`.
Furthermore, menus were allready created for technological interdependencies. To integrate them to the software the following extensions are necessary :

- methods to control the new use-cases in the class `F_Control`.
- call of these methods from the appropriate callback-functions in `Festeval_UG.cpp`.
- extension of the class *`F_Feature`* or the sub-classes with methods to derive the constraints and if necessary to save them in the UG datamodel.
- extension of the method `get_constraints_param` in the class *`F_Feature`* or its sub-classes to store information about the constraints in the `CTransferObject` object.
- extension of the function `Create_Constraints` of the AP224_IM.dll for creation of the constraints in the STEP database.

7.2 Further types of features

In this thesis, the following types of features were implemented :

- *thread (symbolic)*
- *pocket*
- *slot*
- *simple hole*
- *counterbore hole*
- *countersunk hole*
- *planar face*
- *chamfer*

Thereby, the most common types of features are covered. Any further types of features can be implemented if needed. Therefore templates and a methodology was created, which can be found in appendix I. It specifies the steps that have to be done for adding further types of features to the *FESTEVAL* environment.

7.3 Explicit baseshapes

In this thesis, only the implicit types of baseshapes were implemented in the software, but the class-structure was already prepared for the implementation of explicit types of baseshapes. The following Steps are necessary to implement explicit types of baseshapes :

- implementation of the sub-class `F_Explizit_BS` of *`F_BaseShape`* and implementation of all necessary methods.
- extension of the class `F_BaseShapeDialog` with the new type.
- extension of the method `get_baseshape_object` in the class `F_Control` with the new type.
- implementation of a new `Create_*` function in AP224_IM.dll.
- extension of the method `Create_BaseShape` in the class `AP224_IM.`

7.4 Read STEP file

To use product information in Unigraphics which is only available as a STEP file, the capability to read STEP files and convert them into a Unigraphics database is necessary. The menu structure for this functionality was already implemented in this thesis.
Furthermore, the method `init_feature(CTransferObject* step_param)` of the class *`F_Feature`* is already defined to create objects of *`F_Feature`* with parameters from

a STEP database. These methods must be extended for creating the feature in the UG database. The same has to be done for the baseshapes.

To create a feature in Unigraphics, the surfaces for the placement of the feature have to be indicated. This results in two difficulties :

a) the surfaces for the placement are not known and

b) are possibly not existing in the Unigraphics database at this moment.

To solve a), the appropriate information must be stored in the STEP database. The STEP datamodel may have to be extended. Another approach is to determine the surfaces for the placement of the feature by the position. The position is known as a point. The face, on which this point lies, is the placement face for the feature.

To solve b), either a certain chronological order for creating the features must be followed, or a recursive method has to be used to create a parent-feature when needed. For this purpose, the software created in this thesis is storing the features in the STEP file in the same sequence as they were created in Unigraphics. This is done by sorting the list with features (function `F_TOOLS_order_uf_list(..)`) by the *timestamp* of the UG database, which indicates the sequence of creation. If the same sequence is used to create the features in Unigraphics after reading the STEP file, problem b) can not occur.

The method `read_STEPfile` in the class `F_Control` is an attempt for reading STEP files and creating the geometry in the Unigraphics database. To make the method operative, the problems stated above must still be solved.

Appendix

A. Users manual

This is a brief guide to using the *FESTEVAL* design environment which was developed within this thesis. The use of each functionality which was implemented is explained. An overview of the menu structure for the different functionalities with example dialog boxes can be seen in fig. A-1.

A.1 Installation and start

To install the *FESTEVAL* design environment in Unigraphics, the following files have to be copied into the stated destination folders :

file	destination folder
`Festeval.dll`	`UGALLIANCE\site\startup`
`FESTEVAL_UG.men`	`UGALLIANCE\site\startup`
`FESTEVAL_UG.dlg`	`UGALLIANCE\site\application`
`STEP_UG.dlg`	`UGALLIANCE\site\application`

After the next start of Unigraphics, the *FESTEVAL* environment can be launched by choosing *Festeval environment* in the menu *FESTEVAL APPLICATIONS*.

note : Microsoft Visual C++ must be installed on the computer.

A.2 New part

Bevor further functionality can be used, a part must be choosen for treatment in the *FESTEVAL* environment. This part must be opened in Unigraphics.
After choosing *new part*, a feature of the part has to be choosen as a baseshape.

Error messages occur if

a) no part is opened in Unigraphics.
b) the choosen feature is not a valid baseshape.
c) more than one feature was choosen.

Valid baseshapes are

- *block*
- *cylinder*
- *prism*

If a valid baseshape was choosen, all child-features are validated according to *FESTEVAL* specifications.
If a feature is invalid, or its type is not known in *FESTEVAL*, an error message is displayed. If such cases occured, it has to be decided wether these features should be deleted. Only if this is agreed to, the part can be used in the *FESTEVAL* environment and a session is started.

A.3 Create feature

The creation of features is considerably equal to the original Unigraphics functions.
The difference is that the new feature, and all its interacting features, are validated according to *FESTEVAL* specifications immediately after the creation.
If one of those features is invalid, an error message is displayed and the newly created feature is automatically deleted.

A.4 Edit Feature

The feature that is to be edited has to be choosen. In the appearing dialog box, the geometric parameters of the feature can be modified. The feature and its interacting features will automatically be validated after modification. If one of the features is not valid anymore, the modification is undone.

caution : since Edit Feature internally works by deleting and recreating the feature, all child-features will be deleted and the sequence of creation is reordered !

A.5 Delete feature

One or more features can be choosen for deletion.

caution : all child-features will be automatically deleted as well !

A.6 Dimensional Tolerance

For each dimension a tolerance can be set. After choosing this function, a feature must be selected. A dialog box with all the dimensional parameters of the selected feature occurs. After one of the parameters is selected, an upper and a lower value for the tolerance can be entered.
Tolerances will be saved in the Unigraphics file and the STEP exchange file.

A.7 Write STEP file

After the input of a file name, the presently treated part is converted into a STEP database and a physical file according to ISO 10303-21 is created.
This STEP file is the link to the next station in the process chain, where it can be read for further handling.

A.8 Quit *FESTEVAL*

The *FESTEVAL* session will be terminated. To reuse *FESTEVAL* functions, *new part* has to be choosen again to start another session.

A.9 The STEP processor

It is also possible to create STEP files of Unigraphics geometry outside of the *FESTEVAL* environment.

To access this function, *STEP processor* in the menu *FESTEVAL APPLICATIONS* has to be choosen. Like in *new part*, a valid baseshape must be choosen first. The further proceeding complies with *write STEP file*.

If unknown types of features are found, an error message is displayed. These features are not stored in the STEP file.

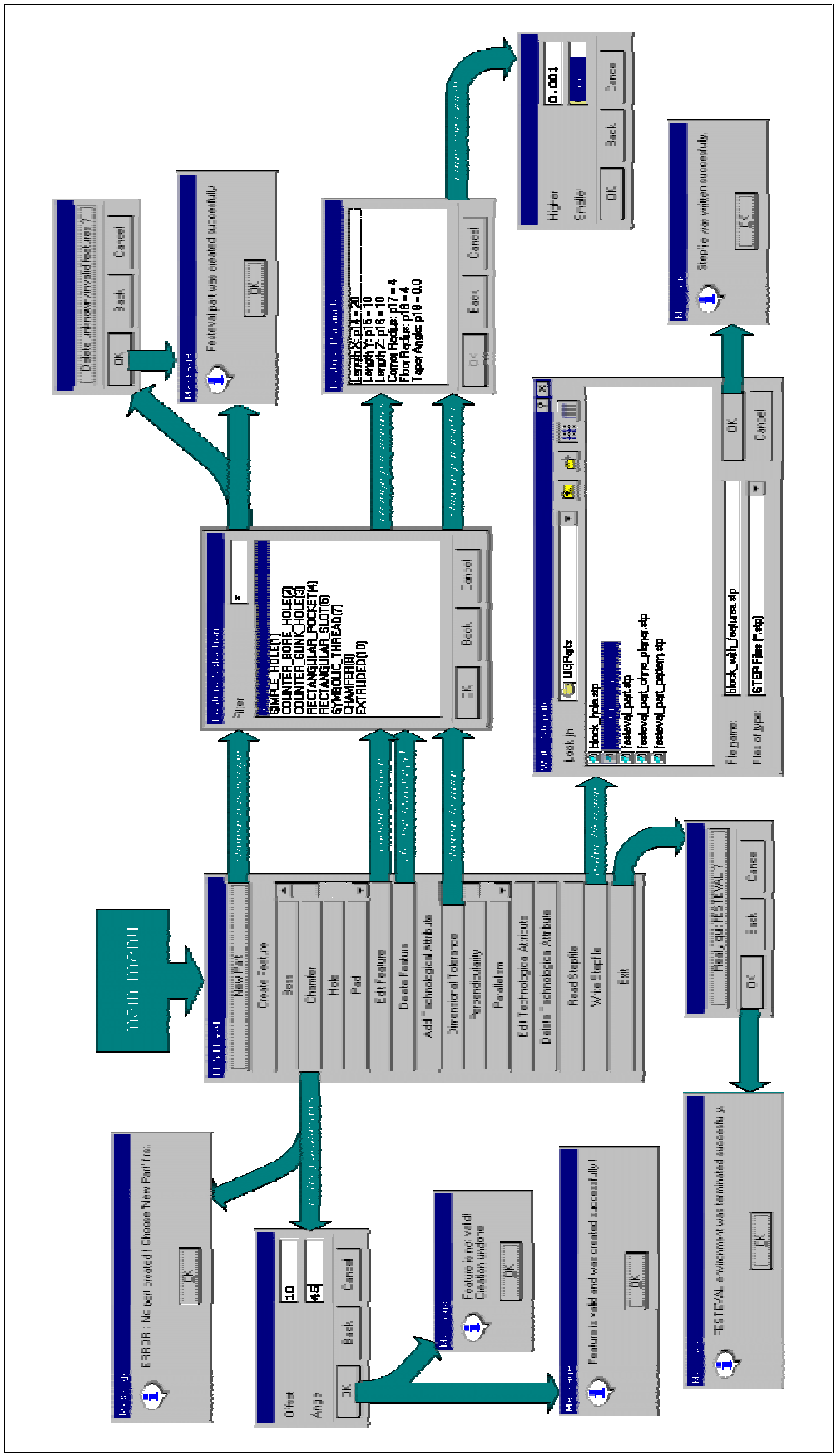


fig. A-1 : menu structure

B. Terms and abbreviations

3D.....3-Dimensional
AAM .....Application Activity Model (part of ->STEP)
AIM .....Application Interpreted Model (part of ->STEP)
AP .....Application Protocol (part of ->STEP)
API.....Application Programmers Interface
ARM .....Application Reference Model (part of ->STEP)
ASCII.....American Standard Code for Information Interchange
CAD.....Computer Aided Design
CD.....Compact Disc
DiK.....Datenverarbeitung in der Konstruktion (institute of ->TUD)
.dll.....Dynamic Link Library
EGE.....Express Graphical Editor
EU.....European Union
database.....data structure used during run-time
datamodel .....abtract structure describing semantics and structure of data
FESTEVAL.....Feature based Support for the Development Process Chain `design-planing-manufacturing´
GUI .....Graphical User Interface
HTML .....Hypertext Markup Language
IMS .....Intelligent Manufacturing Systems
ISO .....International Standards Organization
.lib.....Library (C++)
NIST.....National Institute of Standards and Technology
OMT.....Object Modelling Technique
OO .....Object-Orientation
PDES .....Product Data Exchange Using Step (part of ->NIST)
PTW .....Produktionstechnik und Werkzeugmaschinen (institute of ->TUD)
SCCL .....Step Core Class Library (part of ->SCL)
SCL .....Step Class Library
SCPM.....Loboratório de Sistemas Computacionais para Projeto e Manufatura (institute of ->Unimep)
SDAI.....Standard Data Access Interface (part of ->STEP)
SSCL .....Step Schema Class Libraray (part of ->SCL)
STEP .....Standard for the Exchange of Product Model Data
TUD .....University of Technology Darmstadt
UG .....Unigraphics
UML .....Unified Modeling Language
Unimep.....Universidade Methodista de Piracicaba
USA.....United States of America

C. Index of figures

D. Literature list

D.1 STEP

[And2000] Reiner Anderl / D. Trippner (Hrsg.)
STEP. Standard for the Exchange of Product Model Data
B.G. Teubner 2000

[ISO10303] ISO 10303- 1, 11, 21, 22, 23, 42, 224
ISO Central Secreteriat

D.2 Software development

[Oest97] Bernd Oestereich
Objektorientierte Softwareentwicklung mit der UML, 3. Auflage
Oldenbourg 1997

[Booch2000] Grady Booch
Object-Oriented Analysis and Design
Pearson 2000

[UML99] UML Notation Guide, version 1.3

[Eder2000] Manfred Eder
Umdruck zur Übung Programmiersprachen und –techniken
TU-Darmstadt 2000

[Fow98] M. Fowler / K. Scott
UML destilled
Addison-Wesley 1998

[Lb2000] S. Leibrecht
Einsatz objektorientierter Softwaretechnologie in der technischen Anwendung
TU-Darmstadt 2000

D.3 C++

[Eckel2000] Bruce Eckel
Thinking in C++, second edition
Prentice Hall 2000

[Strou94] Bjarne Stroustrup
The C++ programming language, second edition
Addison-Wesley 1994

[Cub98] Ulrich Cuber
C++ Programmierung – Grundlagen und fortgeschrittene Programmiertechniken
Econ 1998

D.4 Unigraphics

[Ugmn98] Unigraphics User Manuals, Version 15.0
Unigraphics Solutions 1998

[Ugapi98] UG/Open API Reference, Volume 1 – 8, Version 15.0
Unigraphics Solutions 1998

D.5 STEP Class Library

[Lay90] M. McLay / K. Morris
The NIST STEP Class Library (STEP Into The Future)
NISTIR 4411
U.S.Department of commerce 1990

[Mor92] K. Morris / D. Sauder / S. Ressler
Validation Testing System : Reusable Software Component Design
NISTIR 4937
U.S.Department of commerce 1992

[Saud95] A. Sauder / K. Morris
Design of a C++ Software Library for Implementing EXPRESS : The NIST SCL
EXPRESS User Group 1995

[Schzk99] Marcel Schefczik
Konzeption und Implementierung eines Moduls für das Arbeiten mit Kooperationselementen in einem 3D-CAD-System
TU-Darmstadt 1999

D.6 Interdependencies

[Schtz98] Klaus Schützer / Christian Glockner
Integration of machine operator know-how in a feature-based environment
IMS-Europe 1998

[Schtz97] Klaus Schützer
The geometrical and technological interdependencies based on manufacturing features for the integration of systems CAC/CAPP/CAM
Unimep 1997

E. Software list

- Unigraphics V16.0, Unigraphics Solutions Inc. 1999

- UG/Open V.16.0, Unigraphics Solutions Inc. 1999

- Visual C++ 6.0 Professional Edition , Microsoft Corporation 1998

- STEP Class Library 3.1, NIST 1997

- Rational Rose 98 Enterprise Edition, Rational Software Corporation 1998

- Visio Professional 5.0, Visio Corporation 1997

- Micrografx Designer 7, Micrografx Inc. 1997

- Express Graphical Editor 4.0, ProSTEP GmbH 1997

- WinStep 3.0a, Stefan Schwarz 1998

- Netscape Communicator 4.73, Netscape Communications Corp. 2000

- Acrobat Reader 3.02, Adobe Systems Inc. 1997

- WinZip 7.0 SR-1, Nico Mak Computing Inc. 1998

- Office 97, Microsoft Corporation 1997

- Windows NT 4.0 (SP6), Microsoft Corporation 1996

F. Enhancements in the datamodel and libraries

During this thesis the following enhancements have been made in the STEP datamodel and the libraries used to access the STEP datamodel. The enlisted methods, attributes and functions were either newly created (1) or modified (2).

Code that was created or modified in the libraries is not included in Appendix K of this thesis. There, only the code for the *FESTEVAL* design environment is listed. The code of the libraries and the complete STEP datamodel in EGE format can be found on the CD to this thesis. It should be refered to the sourcecode and the comments in it for a more detailed explanation of the enhancements listed here.

F.1 Datamodel

- geometrical parameters for implicit baseshapes (see fig. A-5) :
  (all are of type `numerical_parameter`)

  - `cylindrical_base_shape` : `diameter` (1)

  - `block_base_shape` : `width` (1)
    `height` (1)

  - `ngon_base_shape` : `corner_radius` (1)
    `circumscribed_diameter` (1)
    `number_of_sides` (1)

- changes to constraints (see fig. 3-6) :

  - each constraint has only one `related_feature` (2)
  - name for dependent faces (`face_interaction`) (1)
  - new technological interdependence (`position`) (1)
  - all supertypes are abstract (2)

F.2 Infomodel.lib

- no enhancements

F.3 AP224_IM.dll

- class `CAP224_IM` :	method `AddConstraints`	(1)
	method `ClearModel`	(1)
	method `CreateBaseShape`	(1)
	method `CreateFace`	(1)
	method `CreateFeature`	(2)
	method `IMGetLastIndex`	(1)
	attribute `int last_index`	(1)
- global functions :	function `Create_BlockBaseShape`	(1)
	function `Create_Chamfer`	(2)
	function `Create_Constraints`	(1)
	function `Create_CounterboreHole`	(2)
	function `Create_CountersunkHole`	(1)
	function `Create_CylindricalBaseShape`	(1)
	function `Create_NgonBaseShape`	(1)
	function `Create_PlanarFace`	(1)
	function `Create_RectPocket`	(2)
	function `Create_RoundHole`	(2)
	function `Create_SquareSlot`	(2)
	function `Create_Thread`	(2)
	function `IMAddConstraints`	(1)
	function `IMClearModel`	(1)
	function `IMCreateBaseShape`	(1)
	function `IMCreateFace`	(1)
	function `IMGetLastIndex`	(1)
	function `NumericParameter` `(with tolerances)`	(1)

G. Content of the CD

The CD to this thesis contains everything necessary to continue working on this project except of copyrighted software. Besides the running version of the *FESTEVAL* environment, the sourcecode, libraries and the STEP datamodel this also includes this thesis in digital format, some useful documentations and the used parts in Unigraphics and STEP format.
The directory structure of the CD can be seen in fig. A-2. The directories are containing the following :

Datamodel	files concerning the *FESTEVAL* STEP datamodel
Ap224ext.dex	the STEP datamodel (EGE format)
Html_OLD	older version of the STEP datamodel (HTML format)
Pdf	documentation for AP224 and part42 (PDF format)
PostScript_OLD	older version of the STEP datamodel (PostScript format)
Docu	useful documentations
EckelCPlusPlus	Bruce Eckel's book *Thinking In C++* (PDF format)
Festeval_Schefczik	notes on how to start an UG/Open project (Word format)
SCLDocu	NIST documentation for the SCL (PostScript format)
Uml	UML specifications (PDF format)
UniGraphicsDoc	documentation to Unigraphics & UG/Open (HTML format)
FestevalCode	sourcecode for the *FESTEVAL* environment
Templates	templates for new types of features (-> Appendix I.)
UG_Dialog	dialog and menu files for Unigraphics
VC_Workspace	workspace and project files for Visual C++
PartFiles	parts in Unigraphics and STEP format
RunningVersion	running version of the *FESTEVAL* environment
Thesis	digital version of this thesis (PDF format)
Diagrams	diagrams (Rational Rose, Visio, etc. formats)
ProjectPlan	time schedule of thesis (Excel format)
Word	this thesis (Word format)

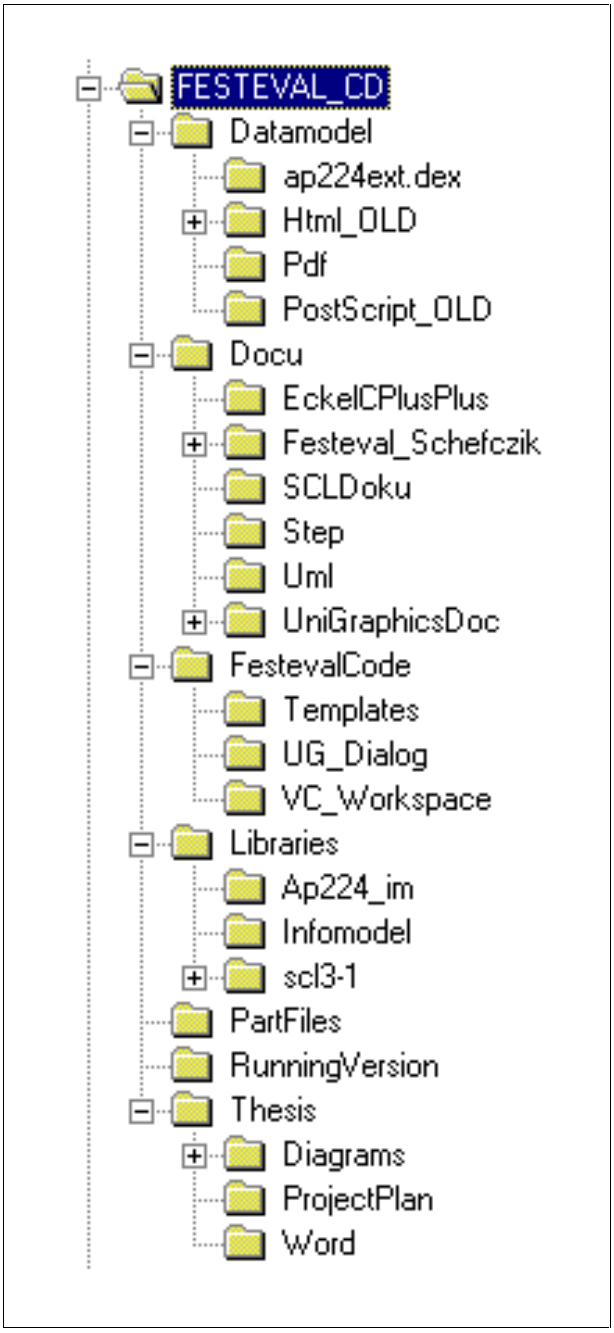


fig. A-2 : content of the CD

H. Express-G diagrams

The most important parts of the STEP datamodel used in this thesis are shown here as Express-G diagrams. See also chapter 2.1.1 and fig. 3-6. The complete STEP datamodel can be found on the CD to this thesis in EGE format.

H.1 Part

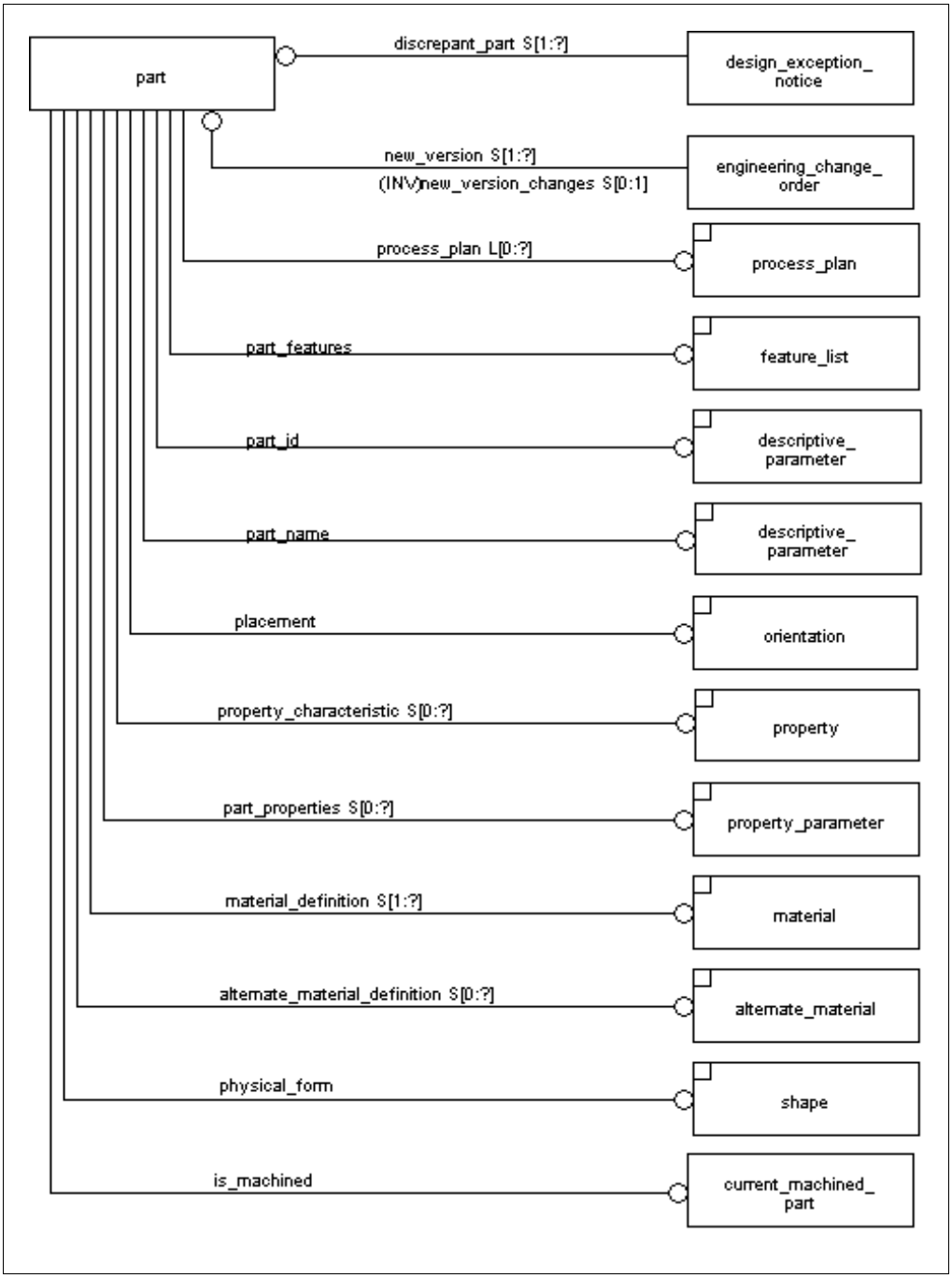


fig. A-3 : Express-G (G.1) – part *(source : DiK)*

H.2 Baseshapes 1

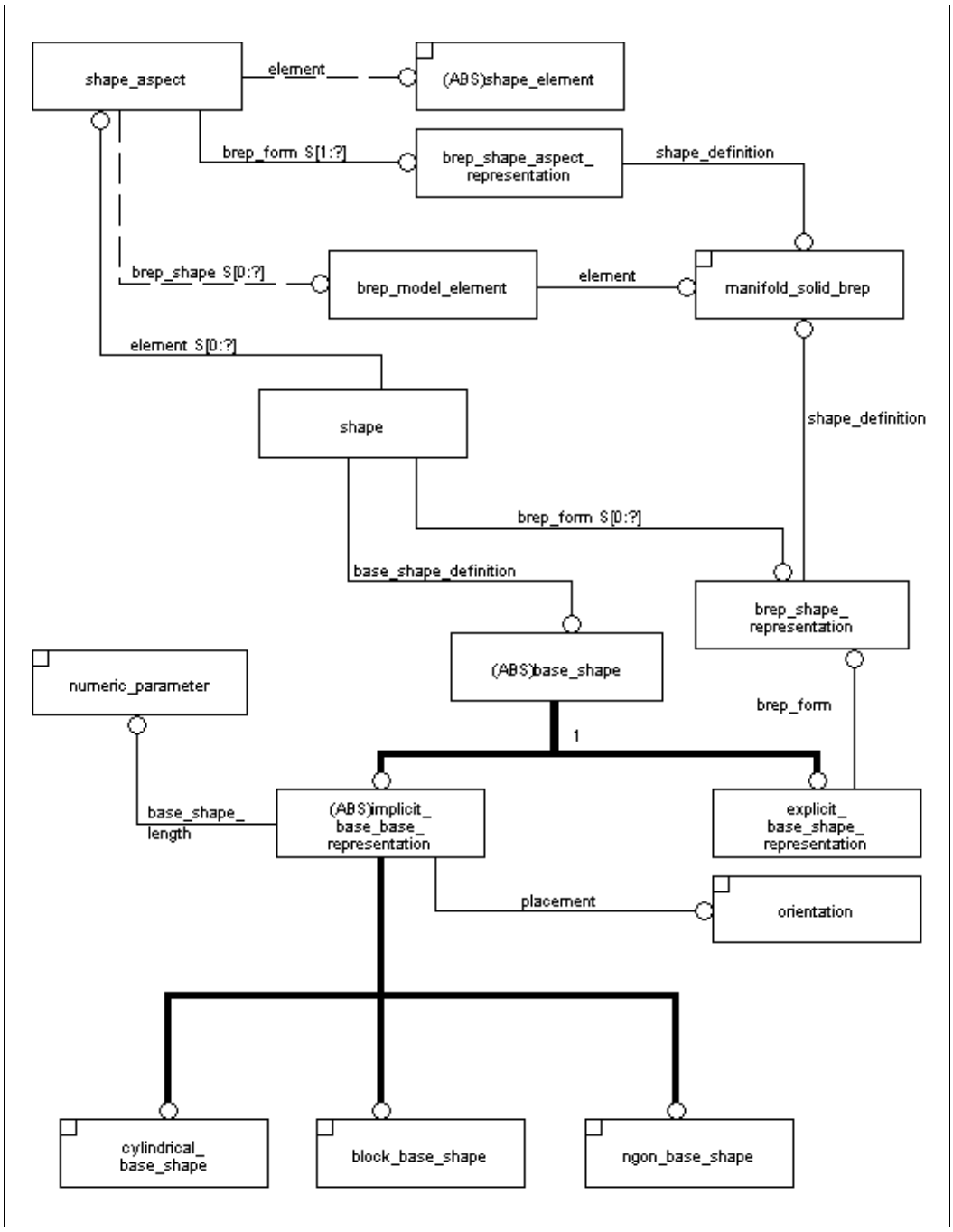


fig. A-4 : Express-G (G.4) – baseshapes 1 *(source : DiK)*

H.3 Baseshapes 2

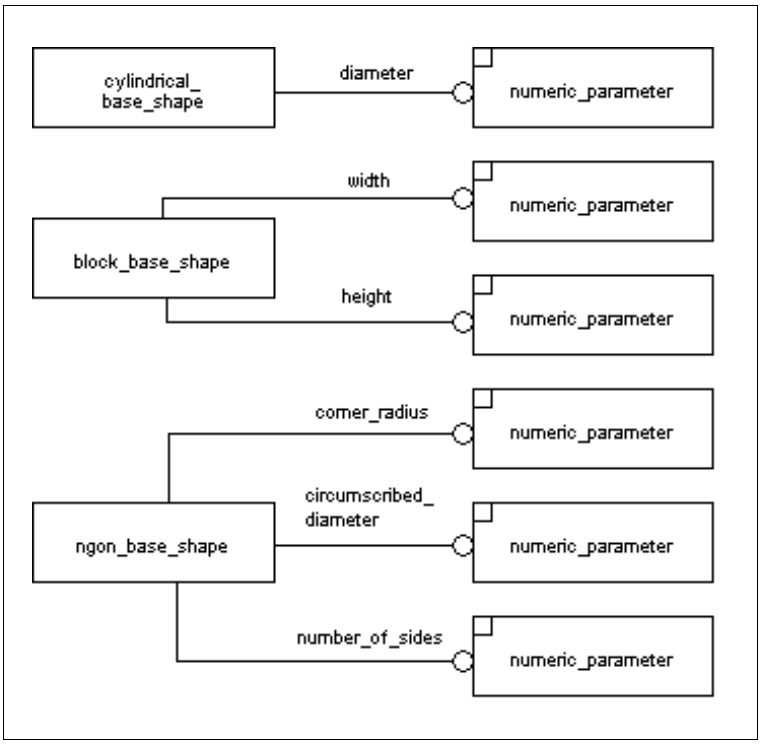


fig. A-5 : Express-G (G.3) – baseshapes 2 *(source : DiK)*

H.4 Feature list

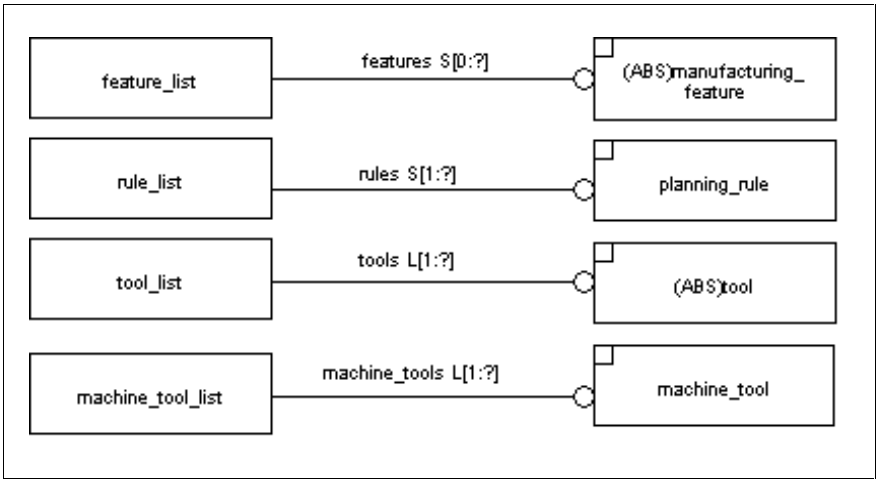


fig. A-6 : Express-G (G.41) – feature list *(source : DiK)*

H.5 Manufacturing feature

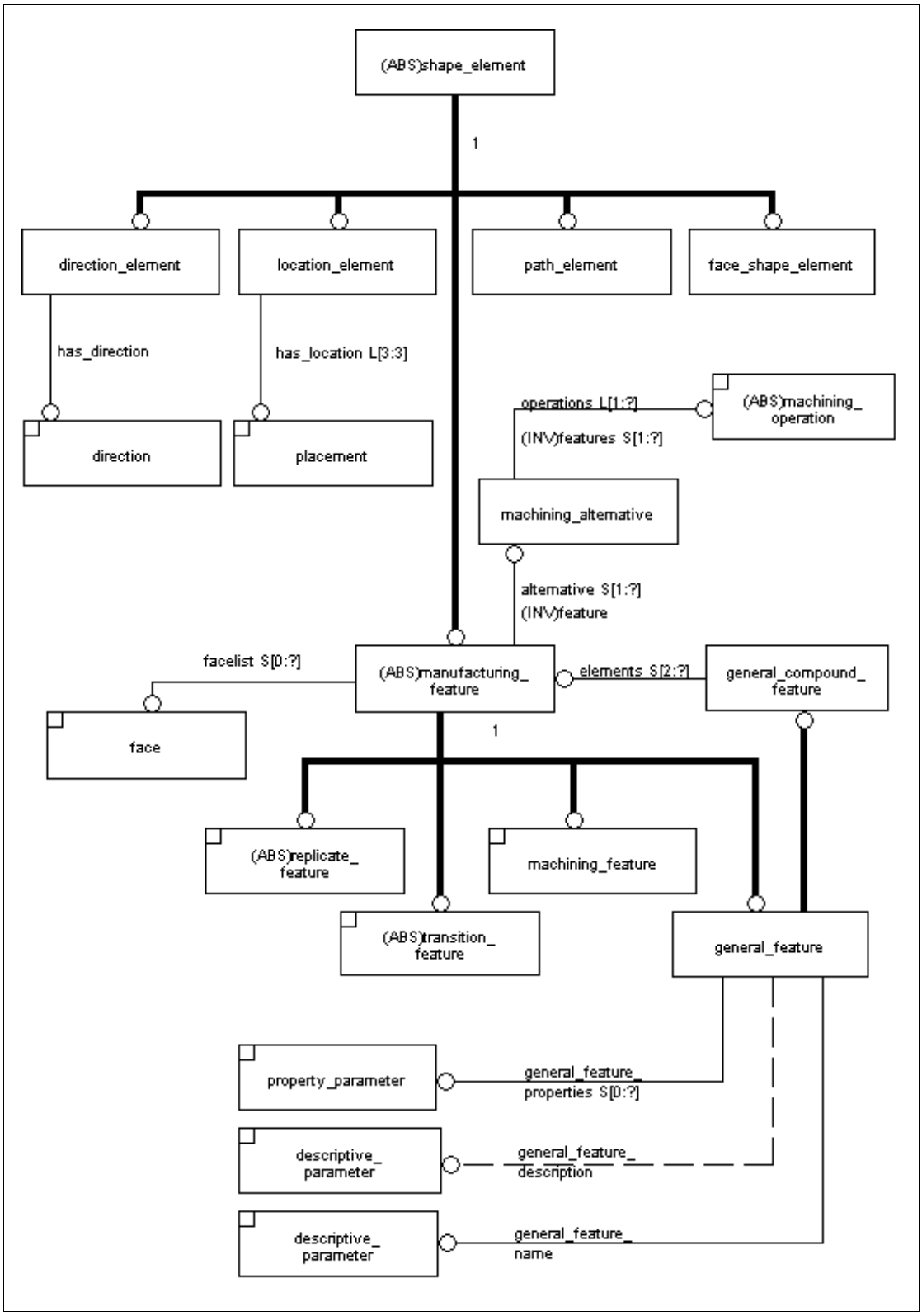


fig. A-7 : Express-G (G.5) – manufacturing feature *(source : DiK)*

H.6 Machining feature

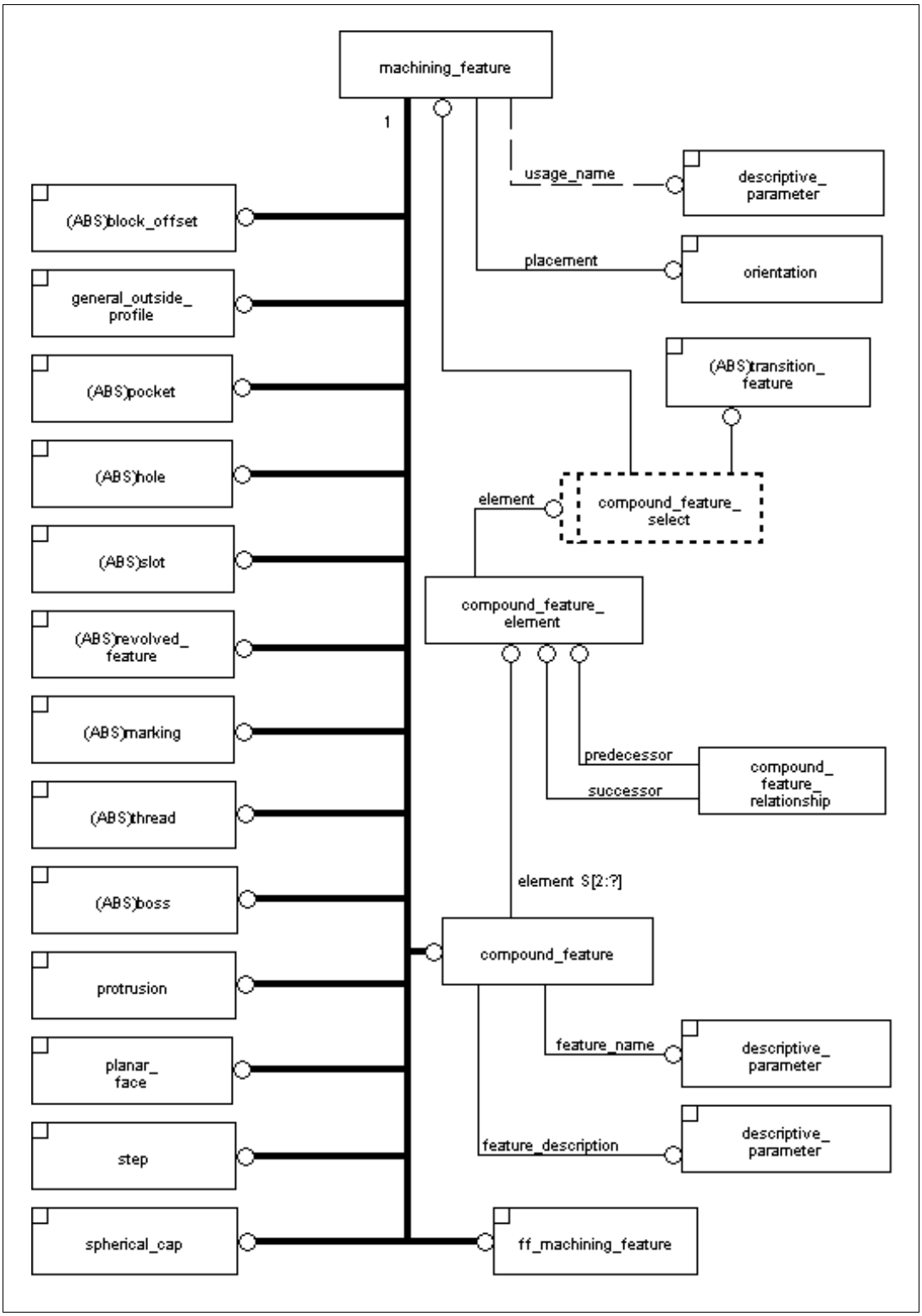


fig. A-8 : Express-G (G.23) – machining feature *(source : DiK)*

I. Methodology for further types of features

The following steps are necessary to add new types of features to the *FESTEVAL* environment. All steps have to be done with the .h.template and the .cpp.template file shown in I.1 and I.2.

1. Save copies of the `.template.*` files with the filnames : `F_Name.cpp` and `F_`*`Name`*`.h` where *Name* is the name of the new feature type, eg. Pocket. Perform the following Steps with the new files !

2. Replace all occurences of `FEATURE_NAME` by the new name of the feature type, eg `Pocket`.

3. Complete the implementation of the methods in the `.cpp` file and delete error messages.

4. Add feature specific methods if needed.

5. Delete unnecessary comments.

6. Add the new files to the project.

7. Add the type for the new feature to `F_Type` in `FESTEVAL_TYPES.h` , eg `F_POCKET`

8. If not done, add a button to `FESTEVAL_UG.dlg` and a callback-function to `FESTEVAL_CB.cpp`. Also add all other necessary information to `FESTEVAL_CB.cpp` and `FESTEVAL_CB.h`.

9. Add the line `festeval_control->create_feature(`*`type`*`);` to the callback-function in `FESTEVAL_CB.cpp`, where *type* is the identifier specified in `F_Type`. Delete the error message if it exists.

10. Add the case for the new feature type to the `F_TOOLS_get_feature_...` methods in `F_TOOLS.cpp`.

11. Include the new `*.h` file in `F_TOOLS.h`.

12. If the new feature type does not exist in the STEP datamodel :
  - Add feature type and parameters to `ap224ext.dex` (with ProSTEP EGE)
  - generate express source code (with ProSTEP EGE)
  - generate C++ classes from sourcecode (with WinStep)
  - Replace classes of `Infomodel.lib` with new ones and recompile `Infomodel.lib`.

13. If feature type is not supported by `AP224.lib` :
  - Create global function `Create_`*FEATURE*`(...)`
    where *FEATURE* is the new feature type.
  - Add case for feature type to the method `CreateFeature` in `AP224_IM.cpp`.

I.1 F_Feature.h.template

```
// F_FEATURE_NAME

/*

Template for a class for a new type of feature to be included in the Festeval-Environment

*/

#ifndef F_FEATURE_NAME_h
#define F_FEATURE_NAME_h

#include "F_Feature.h"

class F_FEATURE_NAME : public F_Feature
{
protected :

// ADDITIONAL ATTRIBUTES NEEDED FOR THIS KIND OF FEATURE

public :

  F_FEATURE_NAME();
  F_FEATURE_NAME(tag_t feature_id);
  F_FEATURE_NAME(CTransferObject step_param);
  CTransferObject* get_step_param();
  bool validate();
  edit();

protected :

  int create_ug_feature();
  int set_face_names();
  int get_faceint_features(tag_t first_face, tag_t second_face, double local_cs[3]);

};

#endif // F_FEATURE_NAME_h
```

I.2 F_Feature.cpp.template

```
// F_FEATURE_NAME

#include "F_FEATURE_NAME.h"
#include "F_Tools.h"


// CONSTRUCTOR (CREATE NEW FEATURE)

F_FEATURE_NAME::F_FEATURE_NAME()
{
  printf("F_FEATURE_NAME\n");

  this->init_feature();
}


// CONSTRUCTOR (FEATURE EXISTS IN UG)

F_FEATURE_NAME::F_FEATURE_NAME(tag_t feature_id)
{
  printf("F_FEATURE_NAME\n");

  this->init_feature(feature_id);
}


// CONSTRUCTOR (FEATURE EXISTS IN STEP)

F_FEATURE_NAME::F_FEATURE_NAME(CTransferObject step_param)
{
  printf("F_FEATURE_NAME\n");

  this->init_feature(step_param);
}


// TRANSFERS AND RETURNS PARAMETERS TO CREATE THE STEP INSTANCE

CTransferObject* F_FEATURE_NAME::get_step_param()
{
  printf("F_FEATURE_NAME::get_step_param\n");

  int retval;
  CTransferObject *param;
  param = F_Feature::get_step_param(); // The feature-unspecific parameters

// (1) Define all the feature-specific parameters
//   that need to be stored in the STEP model

// ADD CODE HERE !!!

// (2) Get all the parameters

// ADD CODE HERE !!!
// USE METHOD UF_MODL_ask_..._parms(...)


// (3) Transfer parameters to STEP parameters

// ADD CODE HERE !!!
// EG. : param->DoubleParam(F_TOOLS_str2dbl(length), "Length");


  return param;

uc1601("ERROR : Function not implemented for F_FEATURE_NAME yet.", 1);
}
```

```
// USER DIALOG TO EDIT FEATURE PARAMETERS

F_FEATURE_NAME::edit()
{
// ADD CODE HERE !!!
uc1601("ERROR : Function not implemented for F_FEATURE_NAME yet.", 1);
}


// USER DIALOG TO CREATE NEW FEATURE

int F_FEATURE_NAME::create_ug_feature()
{
// ADD CODE HERE !!!
uc1601("ERROR : Function not implemented for F_FEATURE_NAME yet.", 1);
return 0;
}


// VALIDATES THE FEATURE

bool F_FEATURE_NAME::validate()
{
// ADD CODE HERE !!!
uc1601("ERROR : Function not implemented for F_FEATURE_NAME yet.", 1);
return true;
}


// SETS THE NAMES OF THE FACES

int F_FEATURE_NAME::set_face_names()
{
// ADD CODE HERE !!!
uc1601("ERROR : Function not implemented for F_FEATURE_NAME yet.", 1);
return 0;
}


// STORES FACE INTERACTING FEATURES IN GLOBAL ATTRIBUTE faceint_features

int   F_FEATURE_NAME::get_faceint_features(tag_t   first_face,   tag_t   second_face,   double
local_cs[3])
{
// ADD CODE HERE !!!
uc1601("ERROR : Function not implemented for F_FEATURE_NAME yet.", 1);
return 0;
}
```

J. Statutory declaration

Hereby I declare that I wrote the thesis at hand independently and without prohibited aids, and only by using the literature stated in the literature list.

Santa Barbara d'Oeste / Brazil, August 11th 2000

S. Leibrecht

K. Sourcecode

The complete sourcecode for the *FESTEVAL* design environment is listed here. The sourcecode for the used libraries is not included.

The implementation of the following methods was mainly handeled by Eng. Nara Gardini of the SCPM, but the sourcecode is included here for the sake of completeness :

```
class F_Control :     create_dimensional_tolerance

class F_Feature :     get_interactions
                      get_interacting_features
                      get_volumeint_features
                      show_volumeint_features
                      show_faceint_features
                      get_edge_features
                      select_location
                      select_direction
                      select_edge
                      get_face_norm
                      get_direction_location
                      create_coord_system

class F_Chamfer :     create_ug_feature
                      edit
                      get_faceint_features
                      set_face_names
                      validate

class F_Hole :        create_ug_feature
                      get_faceint_features
                      set_face_names
                      validate
                      get_hole_diameter

class F_Hole_Cbore :  create_blind_flat_bottom
                      create_blind_tip_angle
                      create_blind_hole
                      create_through_hole
                      edit
```

```
class F_Hole_Csunk :  create_blind_flat_bottom
                      create_blind_tip_angle
                      create_blind_hole
                      create_through_hole
                      edit

class F_Hole_Simple : create_blind_flat_bottom
                      create_blind_tip_angle
                      create_blind_hole
                      create_through_hole
                      edit

class F_Planar_Face : create_ug_feature
                      edit
                      get_faceint_features
                      set_face_names
                      validate

class F_Pocket :      create_ug_feature
                      edit
                      get_faceint_features
                      set_face_names
                      validate
                      get_vertical_edges

class F_Slot :        create_ug_feature
                      edit
                      get_faceint_features
                      set_face_names
                      validate

class F_Thread :      create_ug_feature
                      edit
                      get_faceint_features
                      set_face_names
                      validate
```

K.1 Header files

K.1.1 Festeval_CB.h

```
#ifndef FESTEVAL_UG_H_INCLUDED
#define FESTEVAL_UG_H_INCLUDED

#include <uf.h>
#include <uf_defs.h>
#include <uf_styler.h>

#ifdef __cplusplus
extern "C" {
#endif

#define FESTEVAL_NEW_PART             ("NEW_PART")
#define FESTEVAL_SEP_0                ("SEP_0")
#define FESTEVAL_CREATE_FEATURE       ("CREATE_FEATURE")
#define FESTEVAL_SW_BEG_3             ("SW_BEG_3")
#define FESTEVAL_BOSS                 ("BOSS")
#define FESTEVAL_CHAMFER              ("CHAMFER")
#define FESTEVAL_HOLE                 ("HOLE")
#define FESTEVAL_PAD                  ("PAD")
#define FESTEVAL_PLANAR_FACE          ("PLANAR_FACE")
#define FESTEVAL_POCKET               ("POCKET")
#define FESTEVAL_SLOT                 ("SLOT")
#define FESTEVAL_STEP                 ("STEP")
#define FESTEVAL_THREAD               ("THREAD")
#define FESTEVAL_SW_END_4             ("SW_END_4")
#define FESTEVAL_EDIT_FEATURE         ("EDIT_FEATURE")
#define FESTEVAL_DELETE_FEATURE       ("DELETE_FEATURE")
#define FESTEVAL_SEP_17               ("SEP_17")
#define FESTEVAL_ADD_TECHNOLOGICAL_ATTRIBUTE ("ADD_TECHNOLOGICAL_ATTRIBUTE")
#define FESTEVAL_SW_BEG_22            ("SW_BEG_22")
#define FESTEVAL_DIMENSIONAL_TOLERANCE ("DIMENSIONAL_TOLERANCE")
#define FESTEVAL_FORM_TOLERANCE       ("FORM_TOLERANCE")
#define FESTEVAL_POSITION_TOLERANCE   ("POSITION_TOLERANCE")
#define FESTEVAL_MATERIAL_PROPERTIES  ("MATERIAL_PROPERTIES")
#define FESTEVAL_SW_END_23            ("SW_END_23")
#define FESTEVAL_EDIT_TECHNOLOGICAL_ATTRIBUTE ("EDIT_TECHNOLOGICAL_ATTRIBUTE")
#define FESTEVAL_DELETE_TECHNOLOGICAL_ATTRIBUTE_P_0 ("DELETE_TECHNOLOGICAL_ATTRIBUTE_P_0")
#define FESTEVAL_SEP_2                ("SEP_2")
#define FESTEVAL_READ_STEPFILE        ("READ_STEPFILE")
#define FESTEVAL_WRITE_STEPFILE       ("WRITE_STEPFILE")
#define FESTEVAL_SEP_3                ("SEP_3")
#define FESTEVAL_EXIT                 ("EXIT")
#define FESTEVAL_DIALOG_OBJECT_COUNT  ( 31 )

int FESTEVAL_New_Part_cb ( int dialog_id,
            void * client_data,
            UF_STYLER_item_value_type_p_t callback_data);

int FESTEVAL_Boss_cb ( int dialog_id,
            void * client_data,
            UF_STYLER_item_value_type_p_t callback_data);

int FESTEVAL_Chamfer_cb ( int dialog_id,
            void * client_data,
            UF_STYLER_item_value_type_p_t callback_data);

int FESTEVAL_Hole_cb ( int dialog_id,
            void * client_data,
            UF_STYLER_item_value_type_p_t callback_data);

int FESTEVAL_Pad_cb ( int dialog_id,
            void * client_data,
            UF_STYLER_item_value_type_p_t callback_data);

int FESTEVAL_Planar_Face_cb ( int dialog_id,
            void * client_data,
            UF_STYLER_item_value_type_p_t callback_data);

int FESTEVAL_Pocket_cb ( int dialog_id,
            void * client_data,
            UF_STYLER_item_value_type_p_t callback_data);

int FESTEVAL_Slot_cb ( int dialog_id,
            void * client_data,
            UF_STYLER_item_value_type_p_t callback_data);
```

```
int FESTEVAL_Step_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data);

int FESTEVAL_Thread_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data);

int FESTEVAL_Edit_Feature_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data);

int FESTEVAL_Delete_Feature_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data);

int FESTEVAL_Dimensional_Tolerance_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data);

int FESTEVAL_Form_Tolerance_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data);

int FESTEVAL_Position_Tolerance_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data);

int FESTEVAL_Material_Properties_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data);

int FESTEVAL_Edit_Technological_Attribute_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data);

int FESTEVAL_Delete_Technological_Attribute_act_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data);

int FESTEVAL_Read_Stepfile_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data);

int FESTEVAL_Write_Stepfile_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data);

int FESTEVAL_Exit_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data);


#define STEP_PROC_READ_STEPFILE    ("PROC_READ_STEPFILE")
#define STEP_PROC_WRITE_STEPFILE   ("PROC_WRITE_STEPFILE")

#define STEP_DIALOG_OBJECT_COUNT   ( 2 )


int STEP_Proc_Read_Stepfile_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data);

int STEP_Proc_Write_Stepfile_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data);

#ifdef __cplusplus
}
#endif

#endif /* FESTEVAL_UG_H_INCLUDED */
```

K.1.2 F_Control.h

```
// F_Control

#ifndef F_Control_h
#define F_Control_h

#include <stdio.h>
#include "AP224_IMApi.h"
```

```
#include "FESTEVAL_TYPES.h"
#include "F_Face.h"
#include "F_BaseShape_Dialog.h"
#include "F_BaseShape.h"
#include "F_Block_BS.h"
#include "F_Cylindrical_BS.h"
#include "F_Ngon_BS.h"
#include "F_Feature.h"
#include "F_Tools.h"

class F_Control
{

protected :

  bool part_exists;
  tag_t baseshape_id;
  F_BS_Type baseshape_type;

public :
  F_Control();
  ~F_Control();
  new_part();
  create_feature(F_Type type);
  edit_feature();
  delete_feature();
  create_dimensional_tolerance();
  read_stepfile(bool is_festeval);
  write_stepfile(bool is_festeval);
  exit();

protected :

  F_BaseShape* get_baseshape_object();
  create_step_database();

};

#endif // F_Control_h
```

K.1.3 F_BaseShape_Dialog.h

```
// F_BaseShape_Dialog

#ifndef F_BaseShape_Dialog_h
#define F_BaseShape_Dialog_h
#include "F_BaseShape.h"

class F_BaseShape_Dialog
{
protected :

  tag_t feature_id;
  F_Error status;
  F_BS_Type baseshape_type;

public :

  F_BaseShape_Dialog();
  ~F_BaseShape_Dialog();
  tag_t get_feature_id();
  F_Error get_status();
  F_BS_Type get_type();

};
#endif // F_BaseShape_Dialog_h
```

K.1.4 F_BaseShape.h

```
// F_BaseShape

#ifndef F_BaseShape_h
#define F_BaseShape_h

#include <uf.h>
#include <uf_ui.h>
#include <uf_object_types.h>
#include <uf_modl.h>
#include <uf_vec.h>
#include <uf_mtx.h>
```

```
#include <uf_csys.h>
#include <uf_undo.h>

#include <stdio.h>
#include <ostream.h>
#include <fstream.h>

#include "FESTEVAL_TYPES.h"
#include "AP224_IMApi.h"
#include "TransferObject.h"

#include "ug_api_fkt.h"

class F_BaseShape
{
protected :

  tag_t feature_id;
  F_Error status;

public:

  ~F_BaseShape();
  init_baseshape(tag_t ug_feature_id);
  virtual CTransferObject* get_step_param() = 0;
    F_Error get_status();

};

#endif // F_BaseShape_h
```

K.1.5 F_Implicit_BS.h

```
// F_Implicit_BS

#ifndef F_Implicit_BS_h
#define F_Implicit_BS_h

#include "F_BaseShape.h"

class F_Implicit_BS : public F_BaseShape
{

};

#endif // F_Implicit_BS_h
```

K.1.6 F_Block_BS.h

```
// F_Block_BS

#ifndef F_Block_BS_h
#define F_Block_BS_h

#include "F_Implicit_BS.h"

class F_Block_BS : public F_Implicit_BS
{
public :

  F_Block_BS(tag_t ug_feature_id);
  CTransferObject* get_step_param();

};

#endif // F_Block_BS_h
```

K.1.7 F_Cylindrical_BS.h

```
// F_Cylindrical_BS

#ifndef F_Cylindrical_BS_h
#define F_Cylindrical_BS_h

#include "F_Implicit_BS.h"

class F_Cylindrical_BS : public F_Implicit_BS
{
```

```
public :

  F_Cylindrical_BS(tag_t ug_feature_id);
  CTransferObject* get_step_param();

};

#endif // F_Cylindrical_BS_h
```

K.1.8 F_Ngon_BS.h

```
// F_Ngon_BS

#ifndef F_Ngon_BS_h
#define F_Ngon_BS_h

#include "F_Implicit_BS.h"

class F_Ngon_BS : public F_Implicit_BS
{
public :

  F_Ngon_BS(tag_t ug_feature_id);
  CTransferObject* get_step_param();

};

#endif // F_Ngon_BS_h
```

K.1.9 F_Feature.h

```
// F_Feature

#ifndef F_Feature_h
#define F_Feature_h

#include <uf.h>
#include <uf_ui.h>
#include <uf_object_types.h>
#include <uf_modl.h>
#include <uf_vec.h>
#include <uf_mtx.h>
#include <uf_csys.h>
#include <uf_undo.h>
#include <uf_attr.h>

#include <stdio.h>
#include <ostream.h>
#include <fstream.h>

#include "FESTEVAL_TYPES.h"
#include "F_Face.h"
#include "AP224_IMApi.h"
#include "TransferObject.h"

#include "ug_api_fkt.h"

class F_Feature
{
protected :

  tag_t feature_id;
  bool exists_virtual_face;
  double local_cs[3];
  F_Error status;
  uf_list_p_t interacting_features;

  uf_list_p_t faceint_features;
  uf_list_p_t volumeint_features;

public:

  virtual CTransferObject* get_step_param() = 0;
  CTransferObject* get_constraints_param();
  F_Error get_status();
  ~F_Feature();
  virtual bool validate() = 0;
  uf_list_p_t validate_interacting_features();
  get_dimensional_tolerance();
  virtual edit() = 0;
```

```
  virtual int set_face_names() = 0;
  add_ug_attr_double(char* name, double value);
  double read_ug_attr_double(char* name);
  add_ug_attr_int(char* name, int value);
  int read_ug_attr_int(char* name);

protected :

  init_feature();
  init_feature(tag_t ug_feature_id);
  init_feature(CTransferObject step_param);
  virtual int create_ug_feature() = 0;
  get_interactions();
  virtual uf_list_p_t get_faceint_features() = 0;
  int show_volumeint_features();
  int show_faceint_features();
  int select_location(char *message, tag_t &sel_face_id, double *location, double *face_param);
  int select_direction(char *sel_message, double *direction, tag_t &sel_edge_id);
  int select_edge(char *sel_message, tag_t &edge_id);
  int get_face_norm(tag_t sel_face_id, double *face_param, double *norm);
  int get_direction_location(double dir_x[], double dir_y[], double local_cs[]);
  int create_coord_system(double dir_x[], double dir_y[], double local_cs[], double cross_product[],
tag_t csys_id);
  uf_list_p_t get_edge_features(uf_list_p_t edges_list);
  uf_list_p_t get_interacting_features();
  uf_list_p_t get_volumeint_features();
};

#endif // F_Feature_h
```

K.1.10 F_Chamfer.h

```
// F_Chamfer

#ifndef F_Chamfer_h
#define F_Chamfer_h

#include "F_Feature.h"

class F_Chamfer : public F_Feature
{
protected :

// ADDITIONAL ATTRIBUTES NEEDED FOR THIS KIND OF FEATURE

public :

  F_Chamfer();
  F_Chamfer(tag_t feature_id);
  F_Chamfer(CTransferObject step_param);
  CTransferObject* get_step_param();
  bool validate();
  edit();

protected :

  int create_ug_feature();
  int set_face_names();
  uf_list_p_t get_faceint_features();

};

#endif // F_Chamfer_h
```

K.1.11 F_Hole.h

```
// F_Hole

#ifndef F_Hole_h
#define F_Hole_h

#include "F_Feature.h"

class F_Hole : public F_Feature
{
public :

  bool validate();

protected :
```

```
  int create_ug_feature();
  uf_list_p_t get_faceint_features();
  int get_hole_diameter(char *diameter);

  virtual int create_blind_flat_bottom() = 0;
  virtual int create_blind_tip_angle() = 0;
  virtual int create_blind_hole() = 0;
  virtual int create_through_hole() = 0;
};

#endif // F_Hole_h
```

K.1.12 F_Hole_Simple.h

```
// F_Hole_Simple

#ifndef F_Hole_Simple_h
#define F_Hole_Simple_h

#include "F_Hole.h"

class F_Hole_Simple : public F_Hole
{
public :

  F_Hole_Simple();
  F_Hole_Simple(tag_t feature_id);
  F_Hole_Simple(CTransferObject step_param);
  CTransferObject* get_step_param();
  edit();

protected :
  int set_face_names();

  int create_blind_flat_bottom();
  int create_blind_tip_angle();
  int create_blind_hole();
  int create_through_hole();
};

#endif // F_Hole_Simple_h
```

K.1.13 F_Hole_CBore.h

```
// F_Hole_CBore

#ifndef F_Hole_CBore_h
#define F_Hole_CBore_h

#include "F_Hole.h"

class F_Hole_CBore : public F_Hole
{
public :

  F_Hole_CBore();
  F_Hole_CBore(tag_t feature_id);
  F_Hole_CBore(CTransferObject step_param);
  CTransferObject* get_step_param();
  edit();

protected :
  int set_face_names();

  int create_blind_flat_bottom();
  int create_blind_hole();
  int create_blind_tip_angle();
  int create_through_hole();
};

#endif // F_Hole_CBore_h
```

K.1.14 F_Hole_CSunk.h

```
// F_Hole_CSunk

#ifndef F_Hole_CSunk_h
#define F_Hole_CSunk_h
```

```
#include "F_Hole.h"

class F_Hole_CSunk : public F_Hole
{
public :

  F_Hole_CSunk();
  F_Hole_CSunk(tag_t feature_id);
  F_Hole_CSunk(CTransferObject step_param);
  CTransferObject* get_step_param();
  edit();

protected :
  int set_face_names();

  int create_blind_flat_bottom();
  int create_blind_hole();
  int create_blind_tip_angle();
  int create_through_hole();

};

#endif // F_Hole_CSunk_h
```

K.1.15 F_Planar_Face.h

```
// F_Planar_Face


#ifndef F_Planar_Face_h
#define F_Planar_Face_h

#include "F_Feature.h"

class F_Planar_Face : public F_Feature
{
protected :

// ADDITIONAL ATTRIBUTES NEEDED FOR THIS KIND OF FEATURE

public :

  F_Planar_Face();
  F_Planar_Face(tag_t feature_id);
  F_Planar_Face(CTransferObject step_param);
  CTransferObject* get_step_param();
  bool validate();
  edit();

protected :

  int create_ug_feature();
  int set_face_names();
  uf_list_p_t get_faceint_features();

};

#endif // F_Planar_Face_h
```

K.1.16 F_Pocket.h

```
// F_Pocket

#ifndef F_Pocket_h
#define F_Pocket_h

#include "F_Feature.h"

class F_Pocket : public F_Feature
{
protected :

  double normal_cs[3];

public :

  F_Pocket();
  F_Pocket(tag_t feature_id);
  F_Pocket(CTransferObject step_param);
```

```
  CTransferObject* get_step_param();
  bool validate();
  edit();

protected :

  int create_ug_feature();
  int set_face_names();

  uf_list_p_t get_faceint_features();
  uf_list_p_t get_vertical_edges(uf_list_p_t face_edges);

};

#endif // F_Pocket_h
```

K.1.17 F_Slot.h

```
// F_Slot


#ifndef F_Slot_h
#define F_Slot_h

#include "F_Feature.h"

class F_Slot : public F_Feature
{
protected :

// ADDITIONAL ATTRIBUTES NEEDED FOR THIS KIND OF FEATURE

public :

  F_Slot();
  F_Slot(tag_t feature_id);
  F_Slot(CTransferObject step_param);
  CTransferObject* get_step_param();
  bool validate();
  edit();

protected :

  double normal_cs[3];
  uf_list_p_t get_vertical_edges(uf_list_p_t face_edges);

  int create_ug_feature();
  int set_face_names();
  uf_list_p_t get_faceint_features();

};

#endif // F_Slot_h
```

K.1.18 F_Thread.h

```
// F_Thread


#ifndef F_Thread_h
#define F_Thread_h

#include "F_Feature.h"

class F_Thread : public F_Feature
{
protected :

// ADDITIONAL ATTRIBUTES NEEDED FOR THIS KIND OF FEATURE

public :

  F_Thread();
  F_Thread(tag_t feature_id);
  F_Thread(CTransferObject step_param);
  CTransferObject* get_step_param();
  bool validate();
  edit();

protected :
```

```
  int create_ug_feature();
  int set_face_names();
  uf_list_p_t get_faceint_features();

  int select_cylindrical_face(char *sel_message, tag_t &face_id);
};

#endif // F_Thread_h
```

K.1.19 F_Face.h

```
// F_Face

#ifndef F_Face_h
#define F_Face_h

#include <uf.h>
#include <uf_ui.h>
#include <uf_object_types.h>
#include <uf_modl.h>
#include <uf_vec.h>
#include <uf_mtx.h>
#include <uf_csys.h>
#include <uf_undo.h>
#include <uf_attr.h>

#include <stdio.h>
#include <ostream.h>
#include <fstream.h>

#include "FESTEVAL_TYPES.h"
#include "AP224_IMApi.h"
#include "TransferObject.h"

#include "ug_api_fkt.h"

class F_Face
{
protected :

  tag_t face_id;

public :

  F_Face(tag_t face_id);
  ~F_Face();

  CTransferObject* get_step_param();

  add_ug_attr_double(char* name, double value);
  double read_ug_attr_double(char* name);
  add_ug_attr_int(char* name, int value);
  int read_ug_attr_int(char* name);

protected :

  CStepUG* read_curve(tag_t edge_id, char* name);
  CStepUG* read_point(double point_coords[3], char* name);

};
#endif // F_Face_h
```

K.1.20 F_Tools.h

```
// F_Tools

#include <stdio.h>
#include "AP224_IMApi.h"
#include "FESTEVAL_TYPES.h"
#include "F_Face.h"
#include "F_BaseShape_Dialog.h"
#include "F_BaseShape.h"
#include "F_Block_BS.h"
#include "F_Feature.h"
#include "F_Chamfer.h"
#include "F_Planar_Face.h"
#include "F_Pocket.h"
#include "F_Hole_Simple.h"
#include "F_Hole_CBore.h"
```

```
#include "F_Hole_CSunk.h"
#include "F_Slot.h"
#include "F_Thread.h"

F_Feature* F_TOOLS_get_feature_object(F_Type type);
F_Feature* F_TOOLS_get_feature_object(tag_t feature_id);
F_Feature* F_TOOLS_get_feature_object(CTransferObject* param);
F_Type F_TOOLS_get_feature_type(tag_t feature_id);
F_Type F_TOOLS_get_feature_type(char* type);

double F_TOOLS_str2dbl(const char* string);

uf_list_p_t F_TOOLS_get_feature_list(tag_t base);

uf_list_p_t F_TOOLS_order_uf_list(uf_list_p_t feature_list);
int F_TOOLS_get_time_stamp(tag_t feature_id);

bool F_TOOLS_yes_no_dialog(char* message);
```

K.1.21 Festeval_Types.h

```
// type definitions used in Festeval


#ifndef FESTEVAL_TYPES_H_INCLUDED
#define FESTEVAL_TYPES_H_INCLUDED


// THE TYPES OF FEATURES

enum F_Type {F_UNKNOWN, F_INSTANCE, F_CHAMFER, F_PLANAR_FACE, F_POCKET, F_HOLE_SIMPLE, F_HOLE_CBORE,
F_HOLE_CSUNK, F_SLOT, F_THREAD};


// THE TYPES OF BASESHAPES

enum F_BS_Type {F_BLOCK_BS, F_CYLINDRICAL_BS, F_NGON_BS};


// THE STATUS CODES

enum F_Error {F_OK, F_ERROR, F_CANCELED, F_WORKING, F_NO_PART, F_IO_ERROR, F_VALIDATION_FAILURE,
F_INVALID};


#endif // FESTEVAL_TYPES_H_INCLUDED
```

K.2 Sourcecode files

K.2.1 Festeval_CB.cpp

```
// FESTEVAL_CB

// callback functions for Unigraphics user interface

#include <stdio.h>
#include <uf.h>
#include <uf_defs.h>
#include <uf_exit.h>
#include <uf_ui.h>
#include <uf_styler.h>
#include <uf_mb.h>

#include "FESTEVAL_CB.h"
#include "F_Control.h"
#include "FESTEVAL_TYPES.h"

#include <stdio.h>
#include <uf.h>
#include <uf_defs.h>
#include <uf_exit.h>
#include <uf_ui.h>
#include <uf_styler.h>
#include <uf_mb.h>

#define FESTEVAL_CB_COUNT ( 21 + 1 )
#define STEP_CB_COUNT ( 2 + 1 )
```

```
// THE CONTROL OBJECT : INTERFACE BETWEEN CALLBACK FUNCTIONS AND FESTEVAL OBJECTS
F_Control *festeval_control;

static UF_STYLER_callback_info_t FESTEVAL_cbs[FESTEVAL_CB_COUNT] =
{
 {FESTEVAL_NEW_PART     , UF_STYLER_ACTIVATE_CB     , 1, FESTEVAL_New_Part_cb},
 {FESTEVAL_BOSS         , UF_STYLER_ACTIVATE_CB     , 1, FESTEVAL_Boss_cb},
 {FESTEVAL_CHAMFER      , UF_STYLER_ACTIVATE_CB     , 1, FESTEVAL_Chamfer_cb},
 {FESTEVAL_HOLE         , UF_STYLER_ACTIVATE_CB     , 1, FESTEVAL_Hole_cb},
 {FESTEVAL_PAD          , UF_STYLER_ACTIVATE_CB     , 1, FESTEVAL_Pad_cb},
 {FESTEVAL_PLANAR_FACE  , UF_STYLER_ACTIVATE_CB     , 1, FESTEVAL_Planar_Face_cb},
 {FESTEVAL_POCKET       , UF_STYLER_ACTIVATE_CB     , 1, FESTEVAL_Pocket_cb},
 {FESTEVAL_SLOT         , UF_STYLER_ACTIVATE_CB     , 1, FESTEVAL_Slot_cb},
 {FESTEVAL_STEP         , UF_STYLER_ACTIVATE_CB     , 1, FESTEVAL_Step_cb},
 {FESTEVAL_THREAD       , UF_STYLER_ACTIVATE_CB     , 1, FESTEVAL_Thread_cb},
 {FESTEVAL_EDIT_FEATURE , UF_STYLER_ACTIVATE_CB     , 1, FESTEVAL_Edit_Feature_cb},
 {FESTEVAL_DELETE_FEATURE, UF_STYLER_ACTIVATE_CB     , 1, FESTEVAL_Delete_Feature_cb},
 {FESTEVAL_DIMENSIONAL_TOLERANCE, UF_STYLER_ACTIVATE_CB     , 1, FESTEVAL_Dimensional_Tolerance_cb},
 {FESTEVAL_FORM_TOLERANCE, UF_STYLER_ACTIVATE_CB     , 1, FESTEVAL_Form_Tolerance_cb},
 {FESTEVAL_POSITION_TOLERANCE, UF_STYLER_ACTIVATE_CB     , 1, FESTEVAL_Position_Tolerance_cb},
 {FESTEVAL_MATERIAL_PROPERTIES, UF_STYLER_ACTIVATE_CB     , 1, FESTEVAL_Material_Properties_cb},
 {FESTEVAL_EDIT_TECHNOLOGICAL_ATTRIBUTE,     UF_STYLER_ACTIVATE_CB                          ,     1,
FESTEVAL_Edit_Technological_Attribute_cb},
 {FESTEVAL_DELETE_TECHNOLOGICAL_ATTRIBUTE_P_0,    UF_STYLER_ACTIVATE_CB                      ,     1,
FESTEVAL_Delete_Technological_Attribute_act_cb},
 {FESTEVAL_READ_STEPFILE, UF_STYLER_ACTIVATE_CB     , 1, FESTEVAL_Read_Stepfile_cb},
 {FESTEVAL_WRITE_STEPFILE, UF_STYLER_ACTIVATE_CB     , 1, FESTEVAL_Write_Stepfile_cb},
 {FESTEVAL_EXIT         , UF_STYLER_ACTIVATE_CB     , 1, FESTEVAL_Exit_cb},
 {UF_STYLER_NULL_OBJECT, UF_STYLER_NO_CB, 0, 0 }
};


static UF_STYLER_callback_info_t STEP_cbs[STEP_CB_COUNT] =
{
 {STEP_PROC_READ_STEPFILE, UF_STYLER_ACTIVATE_CB     , 1, STEP_Proc_Read_Stepfile_cb},
 {STEP_PROC_WRITE_STEPFILE, UF_STYLER_ACTIVATE_CB     , 1, STEP_Proc_Write_Stepfile_cb},

 {UF_STYLER_NULL_OBJECT, UF_STYLER_NO_CB, 0, 0 }
};

static UF_MB_styler_actions_t actions[] = {
    { "FESTEVAL_UG.dlg",  NULL,   FESTEVAL_cbs,  UF_MB_STYLER_IS_TOP },
  { "STEP_UG.dlg",  NULL,   STEP_cbs,  UF_MB_STYLER_IS_TOP },
    { NULL,  NULL,  NULL,  0 } /* This is a NULL terminated list */
};

extern void ufsta (char *param, int *retcode, int rlen)
{
    int  error_code;

  FILE *fp;

    if ( (UF_initialize()) != 0)
          return;

  AllocConsole();
  fp = freopen("conout$", "w", stdout);

  printf(" LOG FOR FESTEVAL\n");
  printf(" ----------------\n\n");
  printf("FESTEVAL.DLL was launched !\n");

  festeval_control = new F_Control();

    if ( (error_code = UF_MB_add_styler_actions ( actions ) ) != 0 )
    {
          char fail_message[133];

          UF_get_fail_message(error_code, fail_message);
          printf ( "%s\n", fail_message );
    }

    UF_terminate();
    return;
}


extern int ufusr_ask_unload (void)
{
     return ( UF_UNLOAD_IMMEDIATELY );
}
```

```
extern void ufusr_cleanup (void)
{
    return;
}


int FESTEVAL_New_Part_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

    festeval_control->new_part();

     UF_terminate ();

    return ( UF_UI_CB_CONTINUE_DIALOG );
}


int FESTEVAL_Boss_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

     // festeval_control->create_feature(boss);
  uc1601("ERROR : Feature not implemented yet !", 1);

     UF_terminate ();

    return ( UF_UI_CB_CONTINUE_DIALOG );
}


int FESTEVAL_Chamfer_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

  festeval_control->create_feature(F_CHAMFER);

     UF_terminate ();

    return ( UF_UI_CB_CONTINUE_DIALOG );
}



int FESTEVAL_Hole_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

  char labels[][38] = {"Simple Hole", "Counterbore Hole", "Countersunk Hole"};
  int retval;
  retval = uc1603("Select type of hole", 0, labels, 3);
  if (retval == 5)
  {
    festeval_control->create_feature(F_HOLE_SIMPLE);
  }
  else if (retval == 6)
  {
    festeval_control->create_feature(F_HOLE_CBORE);
  }
  else if (retval == 7)
  {
    festeval_control->create_feature(F_HOLE_CSUNK);
  }


     UF_terminate ();

    return ( UF_UI_CB_CONTINUE_DIALOG );
}
```

```
int FESTEVAL_Pad_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

  //  festeval_control->create_feature(pad);
  uc1601("ERROR : Feature not implemented yet !", 1);

     UF_terminate ();

    return ( UF_UI_CB_CONTINUE_DIALOG );
}



int FESTEVAL_Planar_Face_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

  festeval_control->create_feature(F_PLANAR_FACE);

     UF_terminate ();

    return ( UF_UI_CB_CONTINUE_DIALOG );
}



int FESTEVAL_Pocket_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

  festeval_control->create_feature(F_POCKET);

     UF_terminate ();

    return ( UF_UI_CB_CONTINUE_DIALOG );
}


int FESTEVAL_Slot_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

  festeval_control->create_feature(F_SLOT);

     UF_terminate ();

    return ( UF_UI_CB_CONTINUE_DIALOG );
}



int FESTEVAL_Step_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

//  festeval_control->create_feature(step);
  uc1601("ERROR : Feature not implemented yet !", 1);

     UF_terminate ();

    return ( UF_UI_CB_CONTINUE_DIALOG );
}



int FESTEVAL_Thread_cb ( int dialog_id,
             void * client_data,
```

```
             UF_STYLER_item_value_type_p_t callback_data)
{
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

  festeval_control->create_feature(F_THREAD);

     UF_terminate ();

    return ( UF_UI_CB_CONTINUE_DIALOG );
}


int FESTEVAL_Edit_Feature_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

  festeval_control->edit_feature();

     UF_terminate ();

    return ( UF_UI_CB_CONTINUE_DIALOG );
}


int FESTEVAL_Delete_Feature_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

  festeval_control->delete_feature();

     UF_terminate ();

    return ( UF_UI_CB_CONTINUE_DIALOG );
}



int FESTEVAL_Dimensional_Tolerance_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

  festeval_control->create_dimensional_tolerance();

     UF_terminate ();

    return ( UF_UI_CB_CONTINUE_DIALOG );
}


int FESTEVAL_Form_Tolerance_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

  uc1601("ERROR : Function not implemented yet !", 1);

     UF_terminate ();

    return ( UF_UI_CB_CONTINUE_DIALOG );
}



int FESTEVAL_Position_Tolerance_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

     uc1601("ERROR : Function not implemented yet !", 1);
```

```
     UF_terminate ();

    return ( UF_UI_CB_CONTINUE_DIALOG );
}



int FESTEVAL_Material_Properties_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

     uc1601("ERROR : Function not implemented yet !", 1);

     UF_terminate ();

    return ( UF_UI_CB_CONTINUE_DIALOG );
}


int FESTEVAL_Edit_Technological_Attribute_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

     uc1601("ERROR : Function not implemented yet !", 1);

     UF_terminate ();

    return ( UF_UI_CB_CONTINUE_DIALOG );
}


int FESTEVAL_Delete_Technological_Attribute_act_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

     uc1601("ERROR : Function not implemented yet !", 1);

     UF_terminate ();

    return ( UF_UI_CB_CONTINUE_DIALOG );
}


int FESTEVAL_Read_Stepfile_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

  festeval_control->read_stepfile(true);

     UF_terminate ();

    return ( UF_UI_CB_CONTINUE_DIALOG );
}



int FESTEVAL_Write_Stepfile_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

  festeval_control->write_stepfile(true);

     UF_terminate ();

    return ( UF_UI_CB_CONTINUE_DIALOG );
}
```

```
int FESTEVAL_Exit_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

  festeval_control->exit();

     UF_terminate ();

    return ( UF_UI_CB_CONTINUE_DIALOG );
}


int STEP_Proc_Read_Stepfile_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

   festeval_control->read_stepfile(false);

     UF_terminate ();

   return ( UF_UI_CB_CONTINUE_DIALOG );
}

int STEP_Proc_Write_Stepfile_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

   festeval_control->write_stepfile(false);

     UF_terminate ();

   return ( UF_UI_CB_CONTINUE_DIALOG );
}
```

K.2.2 F_Control.cpp

```
// F_Control

// controls the use-cases

#include "F_Control.h"


// CONSTRUCTOR

F_Control::F_Control()
{
  printf("F_Control\n");

  this->part_exists = false;
}


// DESTRUCTOR

F_Control::~F_Control()
{
  printf("~F_Control\n");
}


// START NEW FESTEVAL SESSION

F_Control::new_part()
{
  if (this->part_exists == true)
  {
    uc1601("ERROR : A part is currently in use. Exit first !", 1);
  }
  else
  {
```

```
// CHOOSE BASESHAPE DIALOG

   F_Error dialog_status;

   F_BaseShape_Dialog* baseshape_dialog = new F_BaseShape_Dialog();

   dialog_status = baseshape_dialog->get_status();

   if (dialog_status == F_OK)
   {
     this->baseshape_id = baseshape_dialog->get_feature_id();
     this->baseshape_type = baseshape_dialog->get_type();

     F_Feature* festeval_feature;

// GET UG FEATURES

     uf_list_p_t feature_list;
     UF_MODL_create_list(&feature_list);

     int feature_count;

     feature_list = F_TOOLS_get_feature_list(this->baseshape_id);
     UF_MODL_ask_list_count(feature_list, &feature_count);

     printf("Number of features : %d", feature_count);
     printf("\n");

     uf_list_p_t del_list;
     UF_MODL_create_list(&del_list);

     int f_nr;
     tag_t feature_id;

// VALIDATE ALL FEATURES EXCEPT BASESHAPE

     for(f_nr = 0; f_nr < feature_count; f_nr++)
     {
       UF_MODL_ask_list_item(feature_list, f_nr, &feature_id);

       if (feature_id != this->baseshape_id)
       {
         festeval_feature = F_TOOLS_get_feature_object(feature_id);

         if (festeval_feature == NULL)
         {
           uc1601("ERROR : Unknown feature found ! Can not be used in FESTEVAL !", 1);
           UF_MODL_put_list_item(del_list, feature_id);
         }
         else
         {
           if (festeval_feature->validate() == FALSE)
           {
             uc1601("ERROR : Invalid feature found ! Can not be used in FESTEVAL !", 1);
             UF_MODL_put_list_item(del_list, feature_id);
           }

           delete festeval_feature;
         }
       }
     }

// WHAT TO DO WITH INVALID FEATURES ?

     int del_count;

     UF_MODL_ask_list_count(del_list, &del_count);

     if (del_count > 0)
     {
       if (F_TOOLS_yes_no_dialog("Delete unknown/invalid features ?") == true)
       {
         UF_MODL_delete_feature(del_list);
         UF_MODL_delete_list(&del_list);
         del_count = 0;
       }
     }

     if (del_count == 0)
     {
       part_exists = true;
       uc1601("Festeval part was created succesfully.", 1);
     }
```

```
      else
      {
        uc1601("ERROR : Unknown or invalid features ! FESTEVAL Part could not be created !", 1);
      }
    }

 // ERRORS FROM CHOOSING A BASESHAPE

    else if (dialog_status == F_INVALID)
    {
      uc1601("ERROR : Choosen baseshape is not valid !", 1);
    }
    else if (dialog_status == F_NO_PART)
    {
      uc1601("ERROR : No UG part to start with available ! Open or create part in UG first !", 1);
    }
    else if (dialog_status == F_ERROR)
    {
      uc1601("ERROR : Only one baseshape can be choosen !", 1);
    }
    delete baseshape_dialog;
  }
}


// CREATE NEW FEATURE

F_Control::create_feature(F_Type type)
{
  F_Feature *festeval_feature;

  if (this->part_exists)
  {
    festeval_feature = F_TOOLS_get_feature_object(type);
    delete festeval_feature;
  }
  else
  {
    uc1601("ERROR : No part created ! Choose 'New Part' first.", 1);
  }
}


// EDIT EXISTING FEATURE

F_Control::edit_feature()
{
  if (this->part_exists)
  {

 // SELECT FEATURES DIALOG

    int count;
    tag_t* feature_tags;
    tag_t feature_id;
    char* feature_type = "";
    int response;
    int retval;

    F_Feature* festeval_feature;

    retval = UF_UI_select_feature ( "Select feature to be edited",
                  NULL,
                  &count,
                  &feature_tags,
                  &response);

    if ((retval == 0) && (response == UF_UI_OK) && (count > 0))
    {
      if (count == 1)
      {
        feature_id = feature_tags[0];

        if (feature_id != this->baseshape_id)
        {

 // CREATE FEATURE OBJECT AND EDIT FEATURE

          festeval_feature = F_TOOLS_get_feature_object(feature_id);
          if (festeval_feature != NULL)
          {
            UF_UNDO_user_visibility_t  visibility = UF_UNDO_any_vis;
            UF_UNDO_mark_name_t    mark_name = NULL;
```

```
            UF_UNDO_mark_id_t      mark_id;
            int validation = UF_UNDO_set_mark(visibility, mark_name, &mark_id);

            festeval_feature->edit();

            if ((festeval_feature->validate() == false) ||
              (festeval_feature->validate_interacting_features() != NULL))
            {
              uc1601("ERROR : New feature or an interacting feature is not valid anymore.*Creation
undone !", 1);
              int validation = UF_UNDO_undo_to_mark(mark_id, mark_name);
            }

            delete festeval_feature;
          }
          else
          {
            uc1601("ERROR : Unknown type of feature. Can not be edited.", 1);
          }
        }
        else
        {
          uc1601("ERROR : Baseshape can not be edited.", 1);
        }
      }
      else
      {
        uc1601("ERROR : Only one feature can be edited at a time.", 1);
      }
    }
  }
  else
  {
    uc1601("ERROR : No part created ! Choose 'New Part' first.", 1);
  }
}


// DELETE EXISTING FEATURE

F_Control::delete_feature()
{
  if (this->part_exists)
  {

 // SELECT FEATURES DIALOG

    int count;
    tag_t* feature_tags;
    tag_t feature_id;
    char* feature_type = "";
    int response;
    int retval;

    uf_list_p_t del_list;

    retval = UF_UI_select_feature ( "Select features to be deleted",
                  NULL,
                  &count,
                  &feature_tags,
                  &response);

    if ((retval == 0) && (response == UF_UI_OK) && (count > 0))
    {

 // CREATING LIST AND ADDING FEATURES

      UF_MODL_create_list(&del_list);
      for (int nr = 0; nr < count; nr++)
      {
        feature_id = feature_tags[nr];
        if (feature_id != this->baseshape_id)
        {
          UF_MODL_put_list_item(del_list, feature_id);
        }
        else
        {
          uc1601("ERROR : Base Shape can not be deleted !", 1);
        }
      }
      if (UF_MODL_delete_feature(del_list) == 0)
      {
        uc1601("Selected features deleted succesfully.", 1);
```

```
      }
      else
      {
        uc1601("WARNING : There were problems while deleting features !", 1);
      }
    }
  }
  else
  {
    uc1601("ERROR : No part created ! Choose 'New Part' first.", 1);
  }
}


// ADD A DIMENSIONAL TOLERANCE TO A FEATURE PARAMAMETER

F_Control::create_dimensional_tolerance()
{
  F_Feature *festeval_feature;

  if (this->part_exists)
  {
    int response = 0;
    int count = 0;
    int retval = 0;
    char message[17] = "Select a feature";
    void * filter = NULL;
    tag_t * feature_tags;
    tag_t feature_id;

    retval = UF_UI_select_feature (
          message,
          filter,
          &count,
          &feature_tags,
          &response);


    if ((retval == 0) && (response == UF_UI_OK))
    {
      if (count == 1)
      {
        feature_id = feature_tags[0];
        if (feature_id != this->baseshape_id)
        {
          festeval_feature = F_TOOLS_get_feature_object(feature_id);
          if (festeval_feature != NULL)
          {
            festeval_feature->get_dimensional_tolerance();
            delete festeval_feature;
          }
          else
          {
            uc1601("ERROR : Unknown type of feature. No tolerances can be added.", 1);
          }
        }
        else
        {
          uc1601("ERROR : No tolerances can be added to Baseshape parameters.", 1);
        }
      }
      else
      {
        uc1601("ERROR : Only one feature can be choosen.", 1);
      }
    }

  }

  else
  {
    uc1601("ERROR : No part created ! Choose 'New Part' first.", 1);
  }
}


// READ STEPFILE AND CREATE UG FEATURES - NOT FINISHED YET !

F_Control::read_stepfile(bool is_festeval)
{
uc1601("ERROR : Function not implemented yet !", 1);
// ADD CODE HERE !!!
```

```
// The following is a basis for reading stepfiles.
// The cunstructor with a parameter of CTransferObject*
// has still to be implemented for each subclass of F_Feature
// and the method init_feature(CTransferObject* param) of the
// abstract superclass F_Feature if necessary.

/*
  if ((this->part_exists == false) || (is_festeval == false))
  {

 // CHOOSE FILENAME

    char mask[132]="*.stp";
    char filename[132] = "";
    int response;

    UF_UI_create_filebox(  "Choose Filename",
                "Write Stepfile",
                mask,
                "d:",
                filename,
                &response);

    if (response == UF_UI_OK)
    {
      int max_id = 0;

      IMClearModel();
      IMReadFile(filename, max_id);

 // GET BASESHAPE AND CREATE UG FEATURE
 // (NOT IMPLEMENTED YET)



 // GET ALL ENTITIES

     int ent_nr;

     for (ent_nr = 1; ent_nr<=max_id; ent_nr++)
     {
       CStepUG* entity;
       CTransferObject* param;
       F_Feature* feature;

       entity = GetStepEntity(ent_nr);

       param = GetFeatureParam(entity);

       UF_UNDO_mark_name_t    mark_name = NULL;
       UF_UNDO_mark_id_t      mark_id;

       UF_UNDO_set_mark(UF_UNDO_any_vis, mark_name, &mark_id);

  // CREATE FESTEVAL OBJECT -> CREATES UG FEATURE AUTOMATICALLY

       feature = F_TOOLS_get_feature_object(param);

       if (feature != NULL)
       {

  // VALIDATE IF FESTEVAL SESSION

         if (is_festeval == true)
         {
           if ((feature->validate() == false) || (feature->validate_interacting_features() != NULL))
           {
             uc1601("ERROR : Invalid feature could not be created.", 1);
             UF_UNDO_undo_to_mark(mark_id, mark_name);
           }
         }
         delete feature;
       }
       delete param;
       delete entity;
     }

     uc1601("File was read succesfully.", 1);
   }
  }
  else
  {
```

```
    uc1601("ERROR : A part is currently in use. Exit first !", 1);
  }

*/
}


// CREATE STEP FILE FROM UG FEATURES

F_Control::write_stepfile(bool is_festeval)
{

// IF NOT FESTEVAL ENVIRONMENT BASESHAPE HAS TO BE CHOOSEN FIRST

  if (is_festeval == false)
  {
    this->part_exists = false;

    F_Error dialog_status;
    F_BaseShape_Dialog* baseshape_dialog = new F_BaseShape_Dialog();
    dialog_status = baseshape_dialog->get_status();

    if (dialog_status == F_OK)
    {
      baseshape_id = baseshape_dialog->get_feature_id();
      baseshape_type = baseshape_dialog->get_type();
      this->part_exists = true;
    }
    else if (dialog_status == F_INVALID)
    {
      uc1601("ERROR : Choosen baseshape is not valid !", 1);
    }
    else if (dialog_status == F_NO_PART)
    {
      uc1601("ERROR : No UG part to start with available ! Open or create part in UG first !", 1);
    }
    else if (dialog_status == F_ERROR)
    {
      uc1601("ERROR : Only one baseshape can be choosen !", 1);
    }
    delete baseshape_dialog;
  }

  if (this->part_exists == true)
  {

 // CHOOSE FILENAME

    char mask[132]="*.stp";
    char filename[132];
    int response;

    UF_UI_create_filebox(  "Choose Filename",
               "Write Stepfile",
               mask,
               "d:",
               filename,
               &response);

    if (response == UF_UI_OK)
    {

 // CREATE STEP MODEL AND WRITE FILE

      IMClearModel();
      this->create_step_database();
      IMWriteFile(filename);
      DeleteInstances();
      uc1601("Stepfile was written succesfully.", 1);
    }
  }
  else if (is_festeval == true)
  {
      uc1601("ERROR : No part created ! Choose 'New Part' first.", 1);
  }

  if (is_festeval == false)
  {
    this->part_exists = false;
  }
}
```

```
// CREATE STEP MODEL

F_Control::create_step_database()
{
  F_BaseShape* new_baseshape;
  CTransferObject* baseshape_param;

  F_Feature* festeval_feature;
  CTransferObject* feature_param;

 // CREATE BASESHAPE

  new_baseshape = get_baseshape_object();

  baseshape_param = new_baseshape->get_step_param();
  IMCreateBaseShape(baseshape_param);
  delete new_baseshape;
  delete baseshape_param;

 // GET UG FEATURES

  uf_list_p_t feature_list;
  UF_MODL_create_list(&feature_list);
  int feature_count;

  feature_list = F_TOOLS_get_feature_list(this->baseshape_id);
  UF_MODL_ask_list_count(feature_list, &feature_count);

  printf("Number of features : %d", feature_count);
  printf("\n");

  int f_nr;
  tag_t feature_id;

 // CREATE FEATURES AND FACES

  for(f_nr = 0; f_nr < feature_count; f_nr++)
  {
    UF_MODL_ask_list_item(feature_list, f_nr, &feature_id);

    if (feature_id != baseshape_id)
    {

  // CREATE F_FEATURE OBJECT

      festeval_feature = F_TOOLS_get_feature_object(feature_id);

      if (festeval_feature == NULL)
      {
        uc1601("ERROR : Unknown feature found ! Can not be converted to STEP !", 1);
      }
      else
      {
        if ((F_TOOLS_get_feature_type(feature_id) != F_THREAD) &&  // CANT ASK FACES FOR THESE
          (F_TOOLS_get_feature_type(feature_id) != F_PLANAR_FACE))
        {
  // CONVERT FACES
          int face_count;
          uf_list_p_t face_list;
          tag_t face_id;
          UF_MODL_ask_feat_faces(feature_id, &face_list);
          UF_MODL_ask_list_count(face_list, &face_count);
          for(int face_nr = 0; face_nr < face_count; face_nr++)
          {
            UF_MODL_ask_list_item(face_list, face_nr, &face_id);
            F_Face* new_face = new F_Face(face_id);
            CTransferObject* face_param = new_face->get_step_param();

            CStepUG* face_entity = IMCreateFace(face_param);

            if (face_entity != NULL)
            {
              new_face->add_ug_attr_int("step index", IMGetLastIndex());
            }

            delete face_param;
            delete new_face;

          }
        }

  // CONVERT FEATURES
```

```
        feature_param = festeval_feature->get_step_param();
        IMCreateFeature(feature_param);
        festeval_feature->add_ug_attr_int("step index", IMGetLastIndex());


        delete festeval_feature;
        delete feature_param;
      }
    }
  }

// ADD CONSTRAINTS

  int step_index;

  for(f_nr = 0; f_nr < feature_count; f_nr++)
  {
    UF_MODL_ask_list_item(feature_list, f_nr, &feature_id);

    if (feature_id != this->baseshape_id)
    {
      festeval_feature = F_TOOLS_get_feature_object(feature_id);

      if (festeval_feature != NULL)
      {
        step_index = festeval_feature->read_ug_attr_int("step index");
        if (step_index > 0)
        {
          feature_param = festeval_feature->get_constraints_param();
          IMAddConstraints(step_index, feature_param);
          delete feature_param;
        }
        delete festeval_feature;
      }
    }
  }
  printf("STEP database was created successfully !\n");
}


// EXIT FESTEVAL SESSION

F_Control::exit()
{
  if (F_TOOLS_yes_no_dialog("Really quit FESTEVAL ?") == true)
  {
    uc1601("FESTEVAL environment was terminated succesfully.", 1);

    if (this->part_exists)
    {
      this->part_exists = false;
    }
  }
}


// GET A FESTEVAL OBJECT FOR THE BASESHAPE

F_BaseShape* F_Control::get_baseshape_object()
{
  F_BaseShape* result = NULL;

  switch(this->baseshape_type)
  {
  case F_BLOCK_BS :

    result = new F_Block_BS(this->baseshape_id);
    break;

  case F_CYLINDRICAL_BS :

    result = new F_Cylindrical_BS(this->baseshape_id);
    break;

  case F_NGON_BS :

    result = new F_Ngon_BS(this->baseshape_id);
    break;
  }

  return result;
}
```

K.2.3 F_BaseShape_Dialog.cpp

```
// F_BaseShape_Dialog

// a dialog for choosing an UG-feature as a festeval-baseshape

#include "F_BaseShape_Dialog.h"


// CONSTRUCTOR

F_BaseShape_Dialog::F_BaseShape_Dialog()
{
  printf("F_BaseShape_Dialog()\n");

  this->status = F_OK;
  this->feature_id = NULL;

  int count;
  tag_t* feature_tags;
  char* feature_type = "";
  int response;
  int retval;

  retval = UF_UI_select_feature ( "Select a base shape for the new part",
                NULL,
                &count,
                &feature_tags,
                &response);

  if (retval == 0)
  {
    if (response == UF_UI_OK)
    {
      if (count != 1)
      {
        this->status = F_ERROR;
      }
      else
      {
        this->feature_id = feature_tags[0];

        UF_MODL_ask_feat_type(this->feature_id, &feature_type);

        printf("Type of feature : %s",feature_type);

        if (strcmp(feature_type, "BLOCK") == 0)
        {
          this->baseshape_type = F_BLOCK_BS;
        }
        else if (strcmp(feature_type, "CYLINDER") == 0)
        {
          this->baseshape_type = F_CYLINDRICAL_BS;
        }
        else if (strcmp(feature_type, "PRISM") == 0)
        {
          this->baseshape_type = F_NGON_BS;
        }

        else
        {
          this->status = F_INVALID;
          this->feature_id = NULL;
        }
      }
    }
    else
    {
      this->status = F_CANCELED;
    }
  }
  else
  {
    this->status = F_NO_PART;
  }
}


// DESTRUCTOR

F_BaseShape_Dialog::~F_BaseShape_Dialog()
{
```

```
  printf("~F_BaseShape_Dialog()\n");
}


// RETURNS THE ID OF THE CHOOSEN BASESHAPE

tag_t F_BaseShape_Dialog::get_feature_id()
{
  return this->feature_id;
}


// RETURNS THE STATUS

F_Error F_BaseShape_Dialog::get_status()
{
  return this->status;
}


// RETURNS THE TYPE OF THE CHOOSEN BASESHAPE

F_BS_Type F_BaseShape_Dialog::get_type()
{
  return this->baseshape_type;
}
```

K.2.4 F_BaseShape.cpp

```
// F_BaseShape

// abstract superclass for all types of baseshapes

#include "F_BaseShape.h"

// DESTRUCTOR

F_BaseShape::~F_BaseShape()
{
  printf("~F_BaseShape\n");
}


// INIT FUNCTION (CALLED BY CONSTRUCTOR)

F_BaseShape::init_baseshape(tag_t ug_feature_id)
{
  this->feature_id = ug_feature_id;
  this->status = F_OK;
}


// RETURNS THE STATUS

F_Error F_BaseShape::get_status()
{
  return this->status;
}


// TRANSFERS AND RETURNS PARAMETERS TO CREATE THE STEP INSTANCE

CTransferObject* F_BaseShape::get_step_param()
{
  int retval;
  CTransferObject *param;
  param = new CTransferObject();

// (1)  Define all the shape-unspecific parameters
//    that need to be stored in the step model

  char*   type;

  double   location[3];
  double   dir_x[3];
  double   dir_y[3];

// (2)  Get all the parameters

  retval = UF_MODL_ask_feat_type(     this->feature_id,
                    &type);
```

```
  retval = UF_MODL_ask_feat_location(   this->feature_id,
                    location);

  retval = UF_MODL_ask_feat_direction(  this->feature_id,
                   dir_x,
                   dir_y);

//  (3) Transfer parameters to step parameters

  param->IntParam(this->feature_id, "Feature_Id");
  param->Type(type);

  param->DoubleArray(location, "Location", 3);
  param->DoubleArray(dir_x, "Dir_x", 3);
  param->DoubleArray(dir_y, "Dir_y", 3);

  return param;
}
```

K.2.5 F_Implicit_BS.cpp

```
// F_Implicit_BS

// abstract superclass for all types of implicit baseshapes

#include "F_Implicit_BS.h"
```

K.2.6 F_Block_BS.cpp

```
// F_Block_BS

// block baseshape

#include "F_Block_BS.h"
#include "F_Tools.h"


// CONSTRUCTOR

F_Block_BS::F_Block_BS(tag_t ug_feature_id)
{
  printf("F_Block_BS()\n");

  this->init_baseshape(ug_feature_id);
}


// TRANSFERS AND RETURNS PARAMETERS TO CREATE THE STEP INSTANCE

CTransferObject* F_Block_BS::get_step_param()
{
  int retval;

  CTransferObject *param;
  param = F_BaseShape::get_step_param(); // The shape-unspecific parameters

// (1)  Define all the shape-specific parameters
//    that need to be stored in the step model

  char* size[3];

// (2)  Get all the parameters

  retval = UF_MODL_ask_block_parms( this->feature_id, 1,
                   size);

//  (3) Transfer parameters to step parameters

  param->DoubleParam(F_TOOLS_str2dbl(size[0]), "Base_Shape_Length");
  param->DoubleParam(F_TOOLS_str2dbl(size[1]), "Width");
  param->DoubleParam(F_TOOLS_str2dbl(size[2]), "Height");

  return param;
}
```

K.2.7 F_Cylindrical_BS.cpp

```
// F_Cylindrical_BS

// cylindrical baseshape

#include "F_Cylindrical_BS.h"
#include "F_Tools.h"


// CONSTRUCTOR

F_Cylindrical_BS::F_Cylindrical_BS(tag_t ug_feature_id)
{
  printf("F_Cylindrical_BS()\n");

  this->init_baseshape(ug_feature_id);
}


// TRANSFERS AND RETURNS PARAMETERS TO CREATE THE STEP INSTANCE

CTransferObject* F_Cylindrical_BS::get_step_param()
{
  int retval;

  CTransferObject *param;
  param = F_BaseShape::get_step_param(); // The shape-unspecific parameters

// (1)  Define all the shape-specific parameters
//    that need to be stored in the step model

  char* diameter;
  char* height;

// (2)  Get all the parameters

  retval = UF_MODL_ask_cylinder_parms(  this->feature_id, 1,
                    &diameter,
                    &height);

//  (3) Transfer parameters to step parameters

  param->DoubleParam(F_TOOLS_str2dbl(height), "Base_Shape_Length");
  param->DoubleParam(F_TOOLS_str2dbl(diameter), "Diameter");

  return param;
}
```

K.2.8 F_Ngon_BS.cpp

```
// F_Ngon_BS

// ngon baseshape

#include "F_Ngon_BS.h"
#include "F_Tools.h"


// CONSTRUCTOR

F_Ngon_BS::F_Ngon_BS(tag_t ug_feature_id)
{
  printf("F_Ngon_BS()\n");

  this->init_baseshape(ug_feature_id);
}


// TRANSFERS AND RETURNS PARAMETERS TO CREATE THE STEP INSTANCE

CTransferObject* F_Ngon_BS::get_step_param()
{
  int retval;

  CTransferObject *param;
  param = F_BaseShape::get_step_param(); // The shape-unspecific parameters

// (1)  Define all the shape-specific parameters
//    that need to be stored in the step model
```

```
  char* diameter;
  char* height;
  char* number_of_sides;

// (2)  Get all the parameters

  retval = UF_MODL_ask_prism_parms( this->feature_id, 1,
                  &diameter,
                  &height,
                  &number_of_sides);

//  (3) Transfer parameters to step parameters

  param->DoubleParam(F_TOOLS_str2dbl(height), "Base_Shape_Length");
  param->DoubleParam(0.0, "Corner_Radius");
  param->DoubleParam(F_TOOLS_str2dbl(diameter), "Circumscribed_Diameter");
  param->DoubleParam(F_TOOLS_str2dbl(number_of_sides), "Number_Of_Sides");

  return param;
}
```

K.2.9 F_Feature.cpp

```
// F_Feature

// abstract superclass for all types of features

#include "F_Feature.h"
#include "F_Tools.h"


// RETURNS STATUS

F_Error F_Feature::get_status()
{
  return this->status;
}


// TRANSFERS AND RETURNS PARAMETERS TO CREATE THE STEP INSTANCE

CTransferObject* F_Feature::get_step_param()
{
  int retval;
  CTransferObject *param;
  tag_t face = NULL;
  param = new CTransferObject();


 // (1) Define all the feature-unspecific parameters
 //   that need to be stored in the step model

  char*   type;

  double    location[3];
  double    dir_x[3];
  double    dir_y[3];

  uf_list_p_t face_list;

 // (2) Get all the parameters


  retval = UF_MODL_ask_feat_type(     this->feature_id,
                    &type);

  retval = UF_MODL_ask_feat_location(   this->feature_id,
                    location);

  retval = UF_MODL_ask_feat_direction(  this->feature_id,
                    dir_x,
                    dir_y);

  retval = UF_MODL_ask_feat_faces(  this->feature_id,
                    &face_list);

 // (3) Transfer parameters to step parameters

  param->IntParam(this->feature_id, "Feature_Id");
  param->Type(type);
```

```
  param->DoubleArray(location, "Location", 3);
  param->DoubleArray(dir_x, "Dir_x", 3);
  param->DoubleArray(dir_y, "Dir_y", 3);

  int face_count;
  tag_t face_id;

  UF_MODL_ask_feat_faces(this->feature_id, &face_list);
  UF_MODL_ask_list_count(face_list, &face_count);
  for(int face_nr = 0; face_nr < face_count; face_nr++)
  {
    int face_index;
    UF_MODL_ask_list_item(face_list, face_nr, &face_id);
    F_Face* face = new F_Face(face_id);
    face_index = face->read_ug_attr_int("step index");
    if (face_index > 0)
    {
 //     param->Reference(GetStepEntity(face_index), "Faces");
 // GetStepEntity (AP224_IM.dll) does not work for faces and most types of features !
    }
    delete face;
  }

  return param;
}


// TRANSFERS AND RETURNS PARAMETERS TO CREATE THE STEP INSTANCES OF THE CONSTRAINTS

CTransferObject* F_Feature::get_constraints_param()
{
  int retval;
  CTransferObject *param;
  tag_t face = NULL;
  param = new CTransferObject();

 // (1) Define all the feature-unspecific parameters
 //   that need to be stored in the step model

 // (2) Get all the parameters

  this->get_interactions();

 // (3) Transfer parameters to step parameters

  int count;
  int f_nr;

 // VOLUME INTERACTIONS

  retval = UF_MODL_ask_list_count(this->volumeint_features, &count);

  for(f_nr = 0; f_nr < count; f_nr++)
  {
    int step_index = 0;
    tag_t f_id;
    F_Feature* int_feature;

    UF_MODL_ask_list_item(volumeint_features, f_nr, &f_id);

    int_feature = F_TOOLS_get_feature_object(f_id);
    step_index = int_feature->read_ug_attr_int("step index");

    if (step_index > 0)
    {
      param->Reference(GetStepEntity(step_index), "VolumeInteractions");
    }

    delete int_feature;
  }

 // FACE INTERACTIONS

  retval = UF_MODL_ask_list_count(this->faceint_features, &count);

  for(f_nr = 0; f_nr < count; f_nr++)
  {
    int step_index = 0;
    tag_t f_id;
    F_Feature* int_feature;

    UF_MODL_ask_list_item(faceint_features, f_nr, &f_id);
```

```
    int_feature = F_TOOLS_get_feature_object(f_id);
    step_index = int_feature->read_ug_attr_int("step index");
    if (step_index > 0)
    {
      param->Reference(GetStepEntity(step_index), "FaceInteractions");
    }

    delete int_feature;
  }

  return param;
}


// INIT FUNCTION (CALLED BY CONSTRUCTOR, CREATE NEW FEATURE)

F_Feature::init_feature()
{
  this->status = F_OK;
  this->interacting_features = NULL;
  this->faceint_features = NULL;
  this->volumeint_features = NULL;

  UF_UNDO_user_visibility_t  visibility = UF_UNDO_any_vis;
  UF_UNDO_mark_name_t    mark_name = NULL;
  UF_UNDO_mark_id_t      mark_id;

  int validation = UF_UNDO_set_mark(visibility, mark_name, &mark_id);

  int retval = this->create_ug_feature(); // create feature
  if (retval==0)
  {
    bool valid = this->validate();  // validate feature

    if (valid == true)
    {
      this->set_face_names();
      this->get_interactions();

      if (this->validate_interacting_features() != NULL)
      {
        valid = false;
      }
    }

    if (valid == false)
    {
      uc1601("Feature is not valid!*Creation undone !", 1);
      int validation = UF_UNDO_undo_to_mark(mark_id, mark_name);
      this->status = F_VALIDATION_FAILURE;
    }
    else
    {
      uc1601("Feature is valid and was created successfully !", 1);
    }
  }
}


// INIT FUNCTION (CALLED BY CONSTRUCTOR, FEATURE EXISTS IN UG)

F_Feature::init_feature(tag_t feature_id)
{
  if (this->feature_id == NULL)
  {
    this->init_feature();
  }
  else
  {
    this->feature_id = feature_id;
    this->status = F_OK;
    this->interacting_features = NULL;
    this->faceint_features = NULL;
    this->volumeint_features = NULL;
  }
}


// INIT FUNCTION (CALLED BY CONSTRUCTOR, FEATURE EXISTS IN STEP)

F_Feature::init_feature(CTransferObject step_param)
{
// CREATE UG FEATURE FROM STEP PARAMS
```

```
// ADD CODE HERE !
this->status = F_OK;
}


// DESTRUCTOR

F_Feature::~F_Feature()
{
  printf("~F_Feature\n");
}


// GETS ALL INTERACTING FEATURES AND STORES THEM IN THE CLASS ATTRIBUTES ...int_features

F_Feature::get_interactions()
{
  if (exists_virtual_face == true)
  {
    this->get_faceint_features();
    this->show_faceint_features();
  }

  this->get_volumeint_features();
  this->show_volumeint_features();
}


// USER DIALOG FOR DIMENSIONAL TOLERANCES

F_Feature::get_dimensional_tolerance()
{
  int array_dim = 2;
  int int_values[] = {0, 0};
  int response = 0;
  int retval = 0;
  int variable_type[] = {301, 301};
  int count = 0;
  char * message = "Select the dimension";
  char * menu_title = "Enter tolerance dimensions";
  char string_array[][16] = {"Upper", "Lower"};
  char value[2][31] = {"0.0000", "0.0000"};
  char * stopstring;
  char * descriptor;
  char * name_upper = (char*) malloc(30);
  char * name_lower = (char*) malloc(30);
  double double_values[] = {0.0, 0.0};
  double tol_value_1;
  double tol_value_2;
  tag_t * expression_tag;

 // USER DIALOG TO SELECT THE PARAMETER

  UF_UI_select_parameters(message, feature_id, &count, &expression_tag, &response);

  if (response == UF_UI_OK && count == 1)
  {
    UF_MODL_ask_descriptor_of_exp(*expression_tag, &descriptor);

 // GET DESCRIPTORS FOR NEW UG ATTRIBUTE

    strcpy(name_upper, descriptor);
    strcat(name_upper, " Upper");

    strcpy(name_lower, descriptor);
    strcat(name_lower, " Lower");

 // GET INITIAL (OLD) VALUES FOR TOLERANCES

    char init_upper[31];
    char init_lower[31];

    _gcvt(this->read_ug_attr_double(name_upper),10, init_upper);
    _gcvt(this->read_ug_attr_double(name_lower),10, init_lower);

    strcpy(value[0], init_upper);
    strcpy(value[1], init_lower);

    strcat(value[0], "0");
    strcat(value[1], "0");

 // USER DIALOG TO ENTER TOLERANCES
```

```
    uc1613(menu_title, string_array, array_dim, int_values, double_values, value, variable_type);

    tol_value_1 = strtod( value[0], &stopstring );
    tol_value_2 = strtod( value[1], &stopstring );

  // CREATE NEW UG ATTRIBUTES WITH TOLRANCE VALUES

    add_ug_attr_double(name_upper, tol_value_1);
    add_ug_attr_double(name_lower, tol_value_2);
  }

  else if (count>1)
  {
    uc1601("ERROR : You can choose just one dimension!", 1);
    return -1;
  }

  else if (response == UF_UI_BACK || response == UF_UI_CANCEL)
  {
    return -1;
  }

  return 0;
}


// RETURNS A LIST WITH INVALID INTERACTING FEATURES

uf_list_p_t F_Feature::validate_interacting_features()
{
  uf_list_p_t invalid_features;
  int count;
  int f_nr;

  UF_MODL_create_list(&invalid_features);

 // VOLUME INTERACTIONS

  UF_MODL_ask_list_count(this->volumeint_features, &count);

  for(f_nr = 0; f_nr < count; f_nr++)
  {
    tag_t f_id;
    F_Feature* int_feature;

    UF_MODL_ask_list_item(this->volumeint_features, f_nr, &f_id);

    int_feature = F_TOOLS_get_feature_object(f_id);

    if (int_feature->validate() == false)
    {
      UF_MODL_put_list_item(invalid_features, f_id);
    }

    delete int_feature;
  }

 // FACE INTERACTIONS

  UF_MODL_ask_list_count(this->faceint_features, &count);

  for(f_nr = 0; f_nr < count; f_nr++)
  {
    tag_t f_id;
    F_Feature* int_feature;

    UF_MODL_ask_list_item(this->faceint_features, f_nr, &f_id);

    int_feature = F_TOOLS_get_feature_object(f_id);

    if (int_feature->validate() == false)
    {
      UF_MODL_put_list_item(invalid_features, f_id);
    }

    delete int_feature;
  }

  UF_MODL_ask_list_count(invalid_features, &count);

  if (count == 0)
  {
    invalid_features = NULL;
```

```
  }

  return invalid_features;
}


// STORES INTERACTING FEATURES IN GLOBAL ATTRIBUTE interacting_features

uf_list_p_t F_Feature::get_interacting_features()
{
  int retval = 0;
  uf_list_p_t edge_list;

  // get list with edges from feature
  retval = UF_MODL_ask_feat_edges (feature_id, &edge_list);
  if (retval!=0) return NULL; // no edges

  interacting_features = get_edge_features(edge_list);

  // deallocate memory
  UF_MODL_delete_list(&edge_list);

  return interacting_features;
}


// STORES VOLUME INTERACTING FEATURES IN GLOBAL ATTRIBUTE volumeint_features

uf_list_p_t F_Feature::get_volumeint_features()
{
  tag_t feature_1, feature_2;
  int count_1, count_2;

  get_interacting_features();

  UF_MODL_ask_list_count(interacting_features, &count_1);

  for(int index_1 = 0; index_1 < count_1; index_1++)
  {
    UF_MODL_ask_list_item(interacting_features, index_1, &feature_1);

    UF_MODL_ask_list_count(faceint_features, &count_2);

    for(int index_2 = 0; index_2 < count_2; index_2++)
    {
      UF_MODL_ask_list_item(faceint_features, index_2, &feature_2);

      if(feature_1 == feature_2)
      {
        UF_MODL_delete_list_item(&interacting_features, feature_1);
      }
    }
  }

  volumeint_features = interacting_features;

  return volumeint_features;
}


// DISPLAYS VOLUME INTERACTING FEATURES

int F_Feature::show_volumeint_features()
{
  tag_t ug_id;
  int   feature_count= 0;
  int   retval     = 0;

//  create output file
  ofstream output("d:\\gardini_n\\show_volumeint_features.html");

//  top of the list
  output << endl;
  output << "<html>\n<head>\n<title>Feature-ID " << feature_id;
  output << ": Validation of features(volume interaction)</title>\n</head>" << endl;
  output << "<body>\n<pre>\n";

  output << "-------------------------------------------------- ------------" << endl;
  output << "       List of volume interaction for Feature " << feature_id  << endl;
  output << "------------------------------------------------------------"<< endl;


  if (volumeint_features != NULL)
```

```
  {
    retval = UF_MODL_ask_list_count(volumeint_features, &feature_count);

    ////////////////////////////////////////////////////////////
    // main data part of the list
    for(int index = 0; index<feature_count; index++) // loop through feature_list
    {
      //  get feature from feature_list
      retval = UF_MODL_ask_list_item(volumeint_features, index, &ug_id);

      ////////////////////////////////////////////////////////
      // put feature_type
      // maybe we can create a function for this!
      char * feature_type_string;
      retval = UF_MODL_ask_feat_type(ug_id, &feature_type_string);

      ////////////////////////////////////////////////////////
      // output section
      // prepare output
      char message_line[79];          // message line for output
      char feature_id_string[10];         // feature_id as a character-string
      char index_string[7];

      _itoa( index, index_string, 5);
      _itoa( ug_id, feature_id_string, 10 ); // put feature_id in the string


      // prepare message_line
      strcpy( message_line, index_string);
      strcat( message_line, " : Feature No. " );
      strcat( message_line, feature_id_string);
      strcat( message_line, " (");
      strcat( message_line, feature_type_string);
      strcat( message_line, ")"); // is " );
  //    if (valid) { strcat( message_line, "valid" );  }
  //    else    { strcat( message_line, "NOT valid" );  }


      // output
      output << message_line << endl;
      // end of output section
      ////////////////////////////////////////////////////////

    } // end of for // loop through feature_list
    // end of main data part of the list
    ////////////////////////////////////////////////////////////

  } // end of if

  else
  {
    output << "there are no feature!" << endl;
  }

  // output bottom of the list
  output << "-----------------------------------------------------------------"<< endl;
  output << "                         End of list"            << endl;
  output << "-----------------------------------------------------------------"<< endl;

  output << "\n</pre>\n</body>\n</html>" << endl;


  // close output file
  output.close();

//  uc1601("Interacting features checked.*See output file for details!", 1);

  return 0;
}


// DISPLAYS FACE INTERACTING FEATURES

int F_Feature::show_faceint_features()
{
  tag_t ug_id;
  int   feature_count= 0;
  int   retval     = 0;

  ofstream output("d:\\gardini_n\\face_interacting.html");

  if (faceint_features != NULL)
  {
```

```
    retval = UF_MODL_ask_list_count(faceint_features, &feature_count);

    output << endl;
    output << "<html>\n<head>\n<title>Feature-ID " << feature_id;
    output << ": Validation of interacting features</title>\n</head>" << endl;
    output << "<body>\n<pre>\n";

    output << "------------------------------------------------ ------------" << endl;
    output << "     List of interacting features for Feature " << feature_id  << endl;
    output << "------------------------------------------------------------"<< endl;


    ////////////////////////////////////////////////////////
    // main data part of the list
    for(int index = 0; index<feature_count; index++) // loop through feature_list
    {
      //  get feature from feature_list
      retval = UF_MODL_ask_list_item(faceint_features, index, &ug_id);

      ////////////////////////////////////////////////////
      // put feature_type
      // maybe we can create a function for this!
      char * feature_type_string;
      retval = UF_MODL_ask_feat_type(ug_id, &feature_type_string);

      ////////////////////////////////////////////////////
      // output section
      // prepare output
      char message_line[79];          // message line for output
      char feature_id_string[10];         // feature_id as a character-string
      char index_string[7];

      _itoa( index, index_string, 5);
      _itoa( ug_id, feature_id_string, 10 ); // put feature_id in the string


      // prepare message_line
      strcpy( message_line, index_string);
      strcat( message_line, " : Feature No. " );
      strcat( message_line, feature_id_string);
      strcat( message_line, " (");
      strcat( message_line, feature_type_string);
      strcat( message_line, ")"); // is " );
  //    if (valid) { strcat( message_line, "valid" );  }
  //    else    { strcat( message_line, "NOT valid" );  }


      // output
      output << message_line << endl;
      // end of output section
      ////////////////////////////////////////////////////


     } // end of for // loop through feature_list
     // end of main data part of the list
     ////////////////////////////////////////////////////////

  }

  else
  {
    output << "there are no feature!" << endl;
  }


  // output bottom of the list
  output << "------------------------------------------------------------"<< endl;
  output << "                         End of list"            << endl;
  output << "------------------------------------------------------------"<< endl;

  output << "\n</pre>\n</body>\n</html>" << endl;


  // close output file
  output.close();

  return 0;
}


// GETS EDGE FEATURES

uf_list_p_t F_Feature::get_edge_features(uf_list_p_t edges_list)
```

```
{
  int edges_list_count = 0;
  int assoc_feature_count = 0;
  int feature_list_count = 0;
  char * Assoc_type_string;
  char *  ug_feature_type_string;
  tag_t Assoc;
  tag_t list_feature;
  bool assoc_in_list = false;
  uf_list_p_t features_list;

  UF_MODL_create_list(&features_list);

  // get number of edges in list
  UF_MODL_ask_list_count(edges_list, &edges_list_count);

  // loop through list of edges
  for (int edges_list_index = 0; edges_list_index < edges_list_count; edges_list_index++)
  {
    // for each edge, check associated features, put in feature_list
    tag_t Edge;
    uf_list_p_t assoc_feature_list;
    UF_MODL_ask_list_item(edges_list, edges_list_index, &Edge);
    UF_MODL_ask_edge_feats(Edge, &assoc_feature_list);

    ////////////////////////////////////////////////////////
    // check, if assoc. features are in list more than one time
    // loop through assoc_feature_list
    // for each assoc.feature, check if it is already in feature_list
    UF_MODL_ask_list_count(assoc_feature_list, &assoc_feature_count);

    // loop through assoc_feature_list
    for(int assoc_list_index = 0; assoc_list_index<assoc_feature_count; assoc_list_index++)
    {
      UF_MODL_ask_list_item(assoc_feature_list, assoc_list_index, &Assoc);

      ////////////////////////////////////////////////////////////
      // loop through features_list; check, if Assoc is in feature_list
      UF_MODL_ask_list_count(features_list, &feature_list_count);

      // loop through feature_list
      for (int feature_list_index = 0; feature_list_index<feature_list_count; feature_list_index++)
      {
        UF_MODL_ask_list_item(features_list, feature_list_index, &list_feature);
        if (list_feature == Assoc)
        {
          assoc_in_list = true;
        }
      } // end of for // loop through feature list

      // check if feature is a block (then no validate)
      UF_MODL_ask_feat_type(Assoc, &Assoc_type_string);
      int not_block = strcmp(Assoc_type_string, "BLOCK");

      if ( (!assoc_in_list) && (feature_id!=Assoc) && (not_block) )
      {
        UF_MODL_put_list_item(features_list, Assoc);
        UF_MODL_ask_feat_type(Assoc, &ug_feature_type_string);
      }

    } // end of for // loop through list of associated features

    // deallocate memory
    UF_MODL_delete_list(&assoc_feature_list);

  } // end of for // loop through list of edges

  return features_list;
}


// USER DIALOG FOR SELECTING A LOCATION

int   F_Feature::select_location(char   *message,   tag_t   &sel_face_id,   double   *location,   double
*face_param)
{
  int  retval = 0;
  int  sel_response = 0;
  tag_t  sel_view_id;
  double  sel_cursor_position[3] = {0., 0., 0.};
  double  * sel_cursor_position_p = sel_cursor_position;
  UF_UI_mask_t  mask;
  UF_UI_selection_options_p_t  sel_options = new UF_UI_selection_options_t;
```

```
  sel_options->other_options        = 0;
  sel_options->reserved           = NULL;
  sel_options->num_mask_triples       = 1;
  sel_options->mask_triples         = &mask;
  sel_options->mask_triples->object_type    = UF_solid_type;
  sel_options->mask_triples->object_subtype = UF_solid_face_subtype;
  sel_options->mask_triples->solid_type   = UF_UI_SEL_FEATURE_PLANAR_FACE;
  sel_options->scope            = UF_UI_SEL_SCOPE_WORK_PART;


  retval = UF_UI_select_single(
           message,
           sel_options,
           &sel_response,
           &sel_face_id,
           sel_cursor_position_p,
           &sel_view_id);


  // check return values from selecting the face

/*  if (retval != 0)
  {
    return -1;
  }
*/
  if  (sel_response == 4 || sel_response == 5)  // 4: selected by name, 5: selected
  {
    retval = UF_DISP_set_highlight(sel_face_id, 0);
  }

  else                     // back or cancel
  {
    return -1;
  }


  delete sel_options;

  double face_point[3] = {0.0, 0.0, 0.0};
  double * face_point_p  = face_point;

  retval = UF_MODL_ask_face_parm(
    sel_face_id,
    sel_cursor_position,
    face_param,
    face_point);

  if (retval != 0)
  {
    return -1;
  }

  // set location
  for (int index=0; index<3; index++) location[index]=face_point[index];

  return 0;
}


// GETS DIRECTION

int F_Feature::select_direction(char *message, double *direction, tag_t &sel_edge_id)
{
  // get edge for direction
  int retval = select_edge(message, sel_edge_id);
  if (retval != 0)
  {
    return -1;
  }

  // get parameters for direction
  int   vertex_count = 0;
  double  point1[3];
  double  point2[3];

  retval = UF_MODL_ask_edge_verts(
    sel_edge_id,
    point1,
    point2,
    &vertex_count);
```

```
  if (retval != 0)
  {
    return -1;
  }


  // set direction
  for (int i=0; i<3; i++) direction[i]=point1[i]-point2[i];

  return 0;

}


// USER DIALOG TO SELECT AN EDGE

int F_Feature::select_edge(char *message, tag_t &edge_id)
{
  int  retval = 0;
  int  sel_response = 0;
  tag_t  sel_view_id;
  double  sel_cursor_position[3] = {0., 0., 0.};
  double  * sel_cursor_position_p = sel_cursor_position;
  UF_UI_mask_t  mask;
  UF_UI_selection_options_p_t  sel_options = new UF_UI_selection_options_t;

  sel_options->other_options        = 0;
  sel_options->reserved           = NULL;
  sel_options->num_mask_triples       = 1;
  sel_options->mask_triples         = &mask;
  sel_options->mask_triples->object_type    = UF_solid_type;
  sel_options->mask_triples->object_subtype = UF_solid_edge_subtype;
  sel_options->mask_triples->solid_type   = UF_UI_SEL_FEATURE_LINEAR_EDGE;
  sel_options->scope            = UF_UI_SEL_SCOPE_WORK_PART;

  retval = UF_UI_select_single(
    message,
    sel_options,
    &sel_response,
    &edge_id,
    sel_cursor_position_p,
    &sel_view_id);



  // check return values from selecting the face

  if  (sel_response == 4 || sel_response == 5)  // 4: selected by name, 5: selected
  {
    retval = UF_DISP_set_highlight(edge_id, 0);
  }

  else                    // back or cancel
  {
    return -1;
  }



  delete sel_options;

  return 0;

}


// GETS THE NORAML OF A FACE

int F_Feature::get_face_norm(tag_t sel_face_id, double *face_param, double *norm)
{
  double point[3]    = {0.0, 0.0, 0.0};
  double unit_normal[3]  = {0.0, 0.0, 0.0};
  double radii[2]    = {0.0, 0.0};
  double dirU1, dirU2, dirV1, dirV2;

  int retval = UF_MODL_ask_face_props(
    sel_face_id,
    face_param,
    point,
    &dirU1, &dirV1,
    &dirU2, &dirV2,
    norm,
    radii);
```

```
  return 0;

}


// GETS DIRECTION / LOCATION

int F_Feature::get_direction_location(double dir_x[], double dir_y[], double local_cs[])
{
  printf("F_Feature::get_direction_location\n");

  int retval = UF_MODL_ask_feat_direction(feature_id, dir_y, dir_x);
  if (retval != 0)
  {
    return -1;
  }

  retval = UF_MODL_ask_feat_location(feature_id, local_cs);
  if (retval != 0)
  {
    return -1;
  }

  return 0;
}


// CREATES COORDINATE SYSTEM

int  F_Feature::create_coord_system(double  dir_x[],  double  dir_y[],  double  local_cs[],  double
cross_product[], tag_t csys_id)
{
  printf("F_Feature::create_coord_system\n");

  double  mtx[9];
  tag_t matrix_id;

  UF_VEC3_cross(dir_x, dir_y, cross_product);
  UF_MTX3_initialize(dir_x, cross_product, mtx);
  UF_CSYS_create_matrix(mtx, &matrix_id);
  UF_CSYS_create_csys(local_cs, matrix_id, &csys_id);

  return 0;
}


// ADDS USER SPECIFIED DOUBLE ATTRIBUTE TO FEATURE

F_Feature::add_ug_attr_double(char* name, double value)
{
  UF_ATTR_value_t uf_value;

  uf_value.type = UF_ATTR_real;
  uf_value.value.real = value;

  UF_ATTR_assign(this->feature_id, name, uf_value);
}


// READS USER SPECIFIED DOUBLE ATRIBUTE FROM FEATURE

double F_Feature::read_ug_attr_double(char* name)
{
  double result;

  UF_ATTR_value_t uf_value;
  int retval;

  retval = UF_ATTR_read_value(this->feature_id, name, UF_ATTR_real, &uf_value);

  if (uf_value.type != 0)
  {
    result = uf_value.value.real;
  }
  else
  {
    result = 0.0;
  }

  return result;
}
```

```
// ADDS USER SPECIFIED INTEGER ATTRIBUTE TO FEATURE

F_Feature::add_ug_attr_int(char* name, int value)
{
  UF_ATTR_value_t uf_value;

  uf_value.type = UF_ATTR_integer;
  uf_value.value.integer = value;

  UF_ATTR_assign(this->feature_id, name, uf_value);
}


// READS USER SPECIFIED INTEGER ATTRIBUTE TO FEATURE

int F_Feature::read_ug_attr_int(char* name)
{
  int result;

  UF_ATTR_value_t uf_value;
  int retval;

  retval = UF_ATTR_read_value(this->feature_id, name, UF_ATTR_integer, &uf_value);

  if (uf_value.type != 0)
  {
    result = uf_value.value.integer;
  }
  else
  {
    result = 0;
  }

  return result;
}
```

K.2.10 F_Chamfer.cpp

```
// F_Chamfer

// chamfer feature

#include "F_Chamfer.h"
#include "F_Tools.h"


// CONSTRUCTOR (CREATE NEW FEATURE)

F_Chamfer::F_Chamfer()
{
  printf("F_Chamfer\n");

  this->init_feature();
}


// CONSTRUCTOR (FEATURE EXISTS IN UG)

F_Chamfer::F_Chamfer(tag_t feature_id)
{
  printf("F_Chamfer\n");

  this->init_feature(feature_id);
}


// CONSTRUCTOR (FEATURE EXISTS IN STEP)

F_Chamfer::F_Chamfer(CTransferObject step_param)
{
  printf("F_Chamfer\n");

  this->init_feature(step_param);
}


// TRANSFERS AND RETURNS PARAMETERS TO CREATE THE STEP INSTANCE

CTransferObject* F_Chamfer::get_step_param()
{
```

```
  int retval;
  CTransferObject *param;
  param = F_Feature::get_step_param(); // The feature-unspecific parameters

// (1)  Define all the feature-specific parameters
//    that need to be stored in the step model

  int   subtype;

  char* radius1;
  double  r1_upper, r1_lower;

  char* radius2;
  double  r2_upper, r2_lower;

  char* theta;
  double  t_upper, t_lower;

// (2)  Get all the parameters

  retval = UF_MODL_ask_chamfer_parms( this->feature_id, 1,
                  &subtype,
                  &radius1,
                  &radius2,
                  &theta);

  r1_upper = this->read_ug_attr_double("Offset 1 Upper");
  r1_lower = this->read_ug_attr_double("Offset 1 Lower");

  r2_upper = this->read_ug_attr_double("Offset 2 Upper");
  r2_lower = this->read_ug_attr_double("Offset 2 Lower");

  t_upper = this->read_ug_attr_double("Theta Upper");
  t_lower = this->read_ug_attr_double("Theta Lower");

//  (3) Transfer parameters to step parameters

  param->DoubleParam(F_TOOLS_str2dbl(radius1), "First_Offset");
  param->DoubleParam(r1_upper, "First_Offset_Upper");
  param->DoubleParam(r1_lower, "First_Offset_Lower");

  param->DoubleParam(F_TOOLS_str2dbl(radius2), "Second_Offset");
  param->DoubleParam(r2_upper, "Second_Offset_Upper");
  param->DoubleParam(r2_lower, "Second_Offset_Lower");

  param->DoubleParam(F_TOOLS_str2dbl(theta), "Chamfer_Angle");
  param->DoubleParam(t_upper, "Chamfer_Angle_Upper");
  param->DoubleParam(t_lower, "Chamfer_Angle_Lower");

  return param;
}


// USER DIALOG TO EDIT FEATURE PARAMETERS

F_Chamfer::edit()
{
  uc1601("ERROR : Function not implemented for F_Chamfer yet.", 1);

// NOT WORKING YET. EDGE FOR CREATION HAS STILL TO BE DETERMINED.

//  int subtype; /*  1 = Single offset.
  //         2 = Double offsets.
  //         3 = Offset and angle.*/
/*  int edit = 1;
  int retval = 0;
  int num_parents = 0;
  int num_children = 0;
  int edges_count = 0;
  int features_count = 0;
  int vertex_count;
  int count = 0;
  char * offset1;
  char * offset2;
  char * angle;
  char * offset1_value;
  char * offset2_value;
  char * angle_value;
  char * exp;
  double point1[3];
  double point2[3];
  double edge_dir[3];
  double dir1[3];
```

```
double dir2[3];
double dir3[3];
double point[3];
tag_t exp_tag;
tag_t * parent_array;
tag_t * children_array;
tag_t block_edge_id;
tag_t chamfer_id;
tag_t edge_feature_id;
tag_t face_id;
tag_t chamfer_edge_id;
uf_list_p_t edge_features;
uf_list_p_t block_edges;
uf_list_p_t feat_list;
uf_list_p_t chamfer_edge;
uf_list_p_t face_list;



UF_MODL_ask_chamfer_parms(feature_id, edit, &subtype, &offset1, &offset2, &angle);

UF_MODL_dissect_exp_string(offset1, &exp, &offset1_value, &exp_tag);
UF_MODL_dissect_exp_string(offset2, &exp, &offset2_value, &exp_tag);
UF_MODL_dissect_exp_string(angle, &exp, &angle_value, &exp_tag);

if (subtype == 1)    // Single offset
{
  printf("\nSUBTYPE:  %d\n", subtype);

  int   array_dimension  = 1;
  int   int_value[]    = { 1 };
  int   variable_type[]  = { 301 };
  char *  chamfer_message   = "Enter chamfer parameters";
  char  menu_list[][16]  = { "Offset" };
  char  string_value[][31] = { "" };
  double  double_value[]   = { 1.0 };

  strcpy(string_value[0], offset1_value);

  retval = uc1613(   // show dialog-box
    chamfer_message,
    menu_list,
    array_dimension,
    int_value,
    double_value,
    string_value,
    variable_type);
  if (retval == 4)
  {
    offset1 = string_value[0];
  }
  else
  {
    return-1;
  }
}


if (subtype == 2)    // Double offsets
{
  printf("\nSUBTYPE:  %d\n", subtype);

  int   array_dimension  = 2;
  int   int_value[]    = { 1, 1 };
  int   variable_type[]  = { 301, 301 };
  char *  chamfer_message   = "Enter chamfer parameters";
  char  menu_list[][16]  = { "Offset1", "Offset2" };
  char  string_value[][31] = { "", "" };
  double  double_value[]   = { 1.0, 1.0 };

  strcpy(string_value[0], offset1_value);
  strcpy(string_value[1], offset2_value);

  retval = uc1613(   // show dialog-box
    chamfer_message,
    menu_list,
    array_dimension,
    int_value,
    double_value,
    string_value,
    variable_type);

  if (retval == 4)
```

```
    {
      offset1 = string_value[0];
      offset2 = string_value[1];
    }
    else
    {
      return-1;
    }
  }

  if (subtype == 3)    // Offset and angle
  {
    printf("\nSUBTYPE:  %d\n", subtype);

    int   array_dimension  = 2;
    int   int_value[]    = { 1, 1 };
    int   variable_type[]  = { 301, 301 };
    char *  chamfer_message    = "Enter chamfer parameters";
    char  menu_list[][16]  = { "Offset", "Angle" };
    char  string_value[][31] = { "", "" };
    double  double_value[]   = { 1.0, 1.0 };

    strcpy(string_value[0], offset1_value);
    strcpy(string_value[1], angle_value);

    retval = uc1613(   // show dialog-box
      chamfer_message,
      menu_list,
      array_dimension,
      int_value,
      double_value,
      string_value,
      variable_type);

    if (retval == 4)
    {
      offset1 = string_value[0];
      angle = string_value[1];
    }
    else
    {
      return-1;
    }
  }


  UF_MODL_ask_feat_relatives(feature_id,       &num_parents,       &parent_array,       &num_children,
&children_array);
  UF_MODL_ask_feat_edges(parent_array[0], &block_edges);
  UF_MODL_ask_list_count(block_edges, &edges_count);

  for(int edge=0; edge<edges_count; edge++)
  {
    UF_MODL_ask_list_item(block_edges, edge, &block_edge_id);
    UF_MODL_ask_edge_feats(block_edge_id, &edge_features);
    UF_MODL_ask_list_count(edge_features, &features_count);

    for(int feature=0; feature<features_count; feature++)
    {
      UF_MODL_create_list(&chamfer_edge);
      UF_MODL_ask_list_item(edge_features, feature, &edge_feature_id);

      if(feature_id == edge_feature_id)
      {
        UF_MODL_ask_edge_faces(edge_feature_id, &face_list);
        UF_MODL_ask_edge_verts(edge_feature_id, point1, point2, &vertex_count);

        edge_dir[0] = point2[0]-point1[0];
        edge_dir[1] = point2[1]-point1[1];
        edge_dir[2] = point2[2]-point1[2];

        if (count == 0)
        {
          dir1[0] = edge_dir[0];
          dir1[1] = edge_dir[1];
          dir1[2] = edge_dir[2];
        }
        else if (count == 1)
        {
          dir2[0] = edge_dir[0];
          dir2[1] = edge_dir[1];
          dir2[2] = edge_dir[2];
        }
```

```
        else if (count == 2)
        {
          dir3[0] = edge_dir[0];
          dir3[1] = edge_dir[1];
          dir3[2] = edge_dir[2];
        }
      }
    }
  }

  UF_MODL_create_list(&feat_list);
  UF_MODL_put_list_item(feat_list, feature_id);
  UF_MODL_delete_feature(feat_list);

  UF_MODL_ask_list_item(face_list, 0, &face_id);

  retval = UF_MODL_ask_extreme(face_id, dir1, dir2, dir3, &chamfer_edge_id, point);
  printf("\nRETVAL EXTREME:  %d\n", retval);

  UF_MODL_create_list(&chamfer_edge);
  UF_MODL_put_list_item(chamfer_edge, chamfer_edge_id);

  UF_DISP_set_highlight(chamfer_edge_id, 1);

  retval = UF_MODL_create_chamfer(subtype, offset1, offset2, angle, chamfer_edge, &chamfer_id);
  printf("\nRETVAL:  %d\n", retval);

  feature_id = chamfer_id;

  return 0;*/
}


// USER DIALOG TO CREATE NEW FEATURE

int F_Chamfer::create_ug_feature()
{
  int subtype = 0; /* Chamfer type:
              1 = Single Offset
              2 = Double Offset
              3 = Offset and Angle */
  int ip2 = 0;
  int array_dimension = 3;
  int response = 0;
  int retval = 0;
  char * message = "Select the type of chamfer";
  char menu_itens[][38] = {"Single Offset", "Double Offset", "Offset and Angle"};
  char * radius1 = "1.000000";
  char * radius2 = "1.000000";
  char * angle = "0.000000";
  char * menu_title = "Enter chamfer parameters";
  tag_t edge_id;
  tag_t feature_id;
  uf_list_p_t edges;


  // Select the type of chamfer
  response = uc1603(message, ip2, menu_itens, array_dimension);

  if(response == 5 || response == 6 || response == 7)
  {
    // Select an edge
    retval = select_edge("Select an edge", edge_id);
    UF_MODL_create_list(&edges);
    UF_MODL_put_list_item(edges, edge_id);

    // Enter parameters
    if(response == 5 && retval == 0)// Single Offset
    {
      int array_dim = 1;
      int int_values[1] = {0};
      int variable_type[1] = {301};
      char string_array[1][16] = {"Offset"};
      char value[1][31] = {"0.0000"};
      double double_values[1] = {0.0};

      subtype = 1;
      uc1613(menu_title, string_array, array_dim, int_values, double_values, value, variable_type);

      radius1 = value[0];
      radius2 = value[0];
    }
```

```
    else if(response == 6 && retval == 0) // Double Offset
    {
      int array_dim = 2;
      int int_values[2] = {0, 0};
      int variable_type[2] = {301, 301};
      char string_array[2][16] = {"Offset 1", "Offset 2"};
      char value[2][31] = {"0.0000", "0.0000"};
      double double_values[2] = {0.0, 0.0};
      subtype = 2;
      uc1613(menu_title, string_array, array_dim, int_values, double_values, value, variable_type);
      radius1 = value[0];
      radius2 = value[1];
    }
    else if (response == 7 && retval == 0)  // Offset and Angle
    {
      int array_dim = 2;
      int int_values[2] = {0, 0};
      int variable_type[2] = {301, 301};
      char string_array[2][16] = {"Offset", "Angle"};
      char value[2][31] = {"0.0000", "0.0000"};
      double double_values[2] = {0.0, 0.0};
      subtype = 3;
      uc1613(menu_title, string_array, array_dim, int_values, double_values, value, variable_type);
      radius1 = value[0];
      angle = value[1];
    }

    else if(retval != 0)
    {
      return -1;
    }
  }
  else if (response == 1 || response == 2)  // back or cancel
  {
    return -1;
  }
  // Create chamfer
  UF_MODL_create_chamfer(subtype, radius1, radius2, angle, edges, &feature_id);

  return 0;
}


// VALIDATES THE FEATURE

bool F_Chamfer::validate()
{
// ADD CODE HERE !!!
return true;
}


// SETS NAMES FOR THE FACES

int F_Chamfer::set_face_names()
{
// ADD CODE HERE !!!
return 0;
}


// STORES FACE INTERACTING FEATURES IN GLOBAL ATTRIBUTE faceint_features

uf_list_p_t F_Chamfer::get_faceint_features()
{
// ADD CODE HERE !!!
return 0;
}
```

K.2.11 F_Hole.cpp

```
// F_Hole

// abstract superclass for all types of holes

#include "F_Hole.h"
#include "F_Tools.h"


// USER DIALOG TO CREATE NEW FEATURE

int F_Hole::create_ug_feature()
```

```
{
  int ip=0;
  char cp[][38] = {"Through", "Blind"};
  int ia[2];
  int retval;

  retval = uc1605("Select hole type", ip, cp, 2, ia);

  if ((retval==1)||(retval==2)) // back or cancel
  {
    printf("\nERROR hole type\n");
    return -1;
  }
  else
  {
    if (ia[0] == 1)
    {
      retval = create_through_hole();
    }
    else if (ia[1] == 1)
    {
      retval = create_blind_hole();
    }
    else
    {
      retval = -1;
    }
  }
  return retval;
}


// VALIDATES FEATURE

bool F_Hole::validate()
{

  // variable declarations
  int     retval;
  int     count;       // number of faces in the hole
  int     cyl_face_count = 0;  // number of cylindrical faces in the hole
  int     face_type;     // type of the face
  tag_t   body_id;     // id of the body which the feature belongs to
  uf_list_p_t face_list;     // list of faces in the feature
  uf_list_t *p_face;     // pointer to first element in face_list

  retval = UF_MODL_ask_feat_body (feature_id, &body_id); // get body of hole
  retval = UF_MODL_ask_feat_faces(feature_id, &face_list);// get all faces of hole
  retval = UF_MODL_ask_list_count(face_list, &count);    // get number of faces

  p_face = face_list;        // set pointer to first element in face_list

  while (p_face != NULL)
  {
    tag_t offset_face;
    tag_t sheet_body_id;
    double  *rp2 = new double;   // offset distance
    double  *rp3 = new double;   // edge curve tolerance
    int   *lp4 = new int;    // not used
//    *rp2 = minimum_wall_size;   // set offset in mm
    *rp2 = 2.0;
    *rp3 = 0.00254;        // set tolerance in mm

    // prepare mark for UNDO after validation
    // ( UNDO the extract and offset process )
    UF_UNDO_user_visibility_t  hole_visibility= UF_UNDO_any_vis;
    UF_UNDO_mark_name_t      hole_mark_name = NULL;
    UF_UNDO_mark_id_t      hole_mark_id;
    int number = UF_UNDO_set_mark(hole_visibility, hole_mark_name, &hole_mark_id);

    retval = UF_MODL_extract_face(p_face->eid, 0, &sheet_body_id);
    FTN(uf5450)(&sheet_body_id, rp2, rp3, lp4, &offset_face);

    retval = UF_MODL_ask_face_type(p_face->eid, &face_type);

    bool valid = false;

    if (face_type == 16 )    //cyl. face
    {
      valid = (0 != UF_MODL_operations (offset_face, body_id,  UF_NEGATIVE));
    }
    else if (face_type == 22) //bounded plane
    {
```

```
      valid = (0 == UF_MODL_operations (offset_face, body_id,  UF_UNSIGNED));
    }
    else if (face_type == 17)
    {
      valid = (0 != UF_MODL_operations (offset_face, body_id,  UF_NEGATIVE));
    }
    else
    {
      printf("\nError with face type in simple hole, type = %d", face_type);
    }

    // UNDO the extract and offset process
    number = UF_UNDO_undo_to_mark(hole_mark_id, hole_mark_name);

    if (!valid)// if 0, then offset_face is not in body => invalid
    {
      return false;// invalid hole
    }

    p_face = p_face->next; // get pointer to next face in face_list
  }

  return true; // valid hole
}


// STORES FACE INTERACTING FEATURES IN GLOBAL ATTRIBUTE faceint_features

uf_list_p_t F_Hole::get_faceint_features()
{
  printf("F_Hole::get_faceint_features\n");

  int face_list_count = 0;
  bool exists_bottom_face = 0;
  bool exists_cylindrical_face = 0;
  bool exists_bore_face = 0;
  bool exists_bore_bottom_face = 0;
  bool exists_csunk_face = 0;
  char face_name[17];
  tag_t bottom_face;
  tag_t cylindrical_face;
  tag_t Face;
  tag_t bore_face;
  tag_t bore_bottom_face;
  tag_t csunk_face;
  uf_list_p_t virtedges_list;
  uf_list_p_t face_list;

  UF_MODL_create_list(&virtedges_list);
  UF_MODL_create_list(&faceint_features);

  UF_MODL_ask_feat_faces(feature_id, &face_list);
  UF_MODL_ask_list_count(face_list, &face_list_count);

  printf("\nFACE_LIST_COUNT:  %d\n", face_list_count);

  for (int face_list_index = 0; face_list_index < face_list_count; face_list_index++)
  {
    UF_MODL_ask_list_item(face_list, face_list_index, &Face);

    int retval = UF_OBJ_ask_name(Face, face_name);

    printf("\nRETVAL:  %d\n", retval);

    if(strcmp(face_name, "BOTTOM_FACE") == 0)
    {
      exists_bottom_face = 1;
      bottom_face = Face;
    }

    else if(strcmp(face_name, "CYLINDRICAL_FACE") == 0)
    {
      exists_cylindrical_face = 1;
      cylindrical_face = Face;
      UF_MODL_ask_face_edges(cylindrical_face, &virtedges_list);
    }

    else if(strcmp(face_name, "BORE_FACE") == 0)
    {
      exists_bore_face = 1;
      bore_face = Face;
      UF_MODL_ask_face_edges(bore_face, &virtedges_list);
    }
```

```
    else if(strcmp(face_name, "BORE_BOTTOM_FACE") == 0)
    {
      exists_bore_bottom_face = 1;
      bore_bottom_face = Face;
    }

    else if(strcmp(face_name, "CSUNK_FACE") == 0)
    {
      exists_csunk_face = 1;
      csunk_face = Face;
      UF_MODL_ask_face_edges(csunk_face, &virtedges_list);
    }
  }


  faceint_features = get_edge_features(virtedges_list);

  return faceint_features;

  return 0;
}


// GETS DIAMETER OF THE HOLE

int F_Hole::get_hole_diameter(char *diameter)
{
  int   retval     = 0;
  char  menu_title[] = "Select hole diameter (mm)";
  int   default_item = 0;
  int   array_dimension= 14;
  char  menu_items[][38]=
  { "5.0",  "6.0",  "7.0",  "8.0",  "9.0",  "10.0", "11.0",
    "12.0", "13.0", "14.0", "15.0", "16.0", "17.0", "18.0"  };

  retval = uc1603(
    menu_title,
    default_item,
    menu_items,
    array_dimension);

  printf("\nget_hole_diameter retval = %d", retval);

  if ((retval==1)||(retval==2)) // back or cancel
  {
    printf("\nERROR hole diameter\n");
    return -1;
  }
  else
  {
    printf("\nstrcpy follows\nmenu_item:  %s", menu_items[retval-5]);
    strcpy(diameter, menu_items[retval-5]);
    printf("\n\ndiameter  = %s", diameter);
  }

  return 0;
}
```

K.2.12 F_Hole_Simple.cpp

```
// F_Hole_Simple

// simle hole feature

#include "F_Hole_Simple.h"
#include "F_Tools.h"


// CONSTRUCTOR (CREATE NEW FEATURE)

F_Hole_Simple::F_Hole_Simple()
{
  printf("F_Hole_Simple\n");
  this->init_feature();
}


// CONSTRUCOR (FEATURE EXISTS IN UG)

F_Hole_Simple::F_Hole_Simple(tag_t feature_id)
{
```

```
  printf("F_Hole_Simple\n");
  this->init_feature(feature_id);
}


// CONSTRUCOTR (FEATURE EXISTS IN STEP)

F_Hole_Simple::F_Hole_Simple(CTransferObject step_param)
{
  printf("F_Hole_Simple\n");
  this->init_feature(step_param);
}


// TRANSFERS AND RETURNS PARAMETERS TO CREATE THE STEP INSTANCE

CTransferObject* F_Hole_Simple::get_step_param()
{
  int retval;
  CTransferObject *param;
  param = F_Feature::get_step_param(); // The feature-unspecific parameters

// (1)  Define all the feature-specific parameters
//    that need to be stored in the step model

  char*   diameter;
  double    di_upper, di_lower;

  char*   depth;
  double    dp_upper, dp_lower;

  char*   tip_angle;
  double    ta_upper, ta_lower;

  int     thru_flag;

// (2)  Get all the parameters

  retval = UF_MODL_ask_simple_hole_parms( this->feature_id, 1,
                    &diameter,
                    &depth,
                    &tip_angle,
                    &thru_flag);

  di_upper = this->read_ug_attr_double("Diameter Upper");
  di_lower = this->read_ug_attr_double("Diameter Lower");

  dp_upper = this->read_ug_attr_double("Depth Upper");
  dp_lower = this->read_ug_attr_double("Depth Lower");

  ta_upper = this->read_ug_attr_double("Tip Angle Upper");
  ta_lower = this->read_ug_attr_double("Tip Angle Lower");

//  (3) Transfer parameters to step parameters

  param->DoubleParam(F_TOOLS_str2dbl(diameter), "Diameter");
  param->DoubleParam(di_upper, "Diameter_Upper");
  param->DoubleParam(di_lower, "Diameter_Lower");

  if (! thru_flag)
  {
    param->DoubleParam(F_TOOLS_str2dbl(depth), "Depth");
    param->DoubleParam(dp_upper, "Depth_Upper");
    param->DoubleParam(dp_lower, "Depth_Lower");
  }

  param->DoubleParam(F_TOOLS_str2dbl(tip_angle), "TipAngle");
  param->DoubleParam(ta_upper, "TipAngle_Upper");
  param->DoubleParam(ta_lower, "TipAngle_Lower");

  param->IntParam(thru_flag, "ThroughBottom");

  param->DoubleParam(0.0, "EdgeRadius");

  return param;
}


// USER DIALOG TO EDIT FEATURES PARAMETERS

F_Hole_Simple::edit()
{
  int retval = 0;
```

```
int edit = 1;
int num_parents;
int num_children;
int faces_count = 0;
int thru_flag = false;
char * diameter;
char * depth = NULL;
char * tip_angle = NULL;
char * left;
char * diameter_value;
char * depth_value;
char * tip_angle_value;
double direction[3];
double location[3];
double dir_x[3];
double dir_z[3];
double face_param[2];
double norm[3];
tag_t hole_id;
tag_t exp_tag;
tag_t face_id;
tag_t * parent_array;
tag_t * children_array;
tag_t block_face_id;
tag_t face_thru = NULL_TAG;
uf_list_p_t feat_list;
uf_list_p_t block_faces;

UF_MODL_ask_simple_hole_parms(feature_id, edit, &diameter, &depth, &tip_angle, &thru_flag);

UF_MODL_dissect_exp_string(diameter, &left, &diameter_value, &exp_tag);
UF_MODL_dissect_exp_string(depth, &left, &depth_value, &exp_tag);
UF_MODL_dissect_exp_string(tip_angle, &left, &tip_angle_value, &exp_tag);

if(thru_flag == 1) // thru hole
{
  printf("\nTHRU HOLE\n");

  int  array_dimension = 1;
  int  int_value[] = { 1 };
  int  variable_type[] = { 301 };
  char * hole_message = "Enter simple hole parameters";
  char menu_list[][16] = { "Diameter" };
  char string_value[][31]  = { "" };
  double double_value[] = { 1.0 };

  strcpy(string_value[0], diameter_value);

  retval = uc1613(   // show dialog-box
    hole_message,
    menu_list,
    array_dimension,
    int_value,
    double_value,
    string_value,
    variable_type);

  if (retval == 4)
  {
    diameter  = string_value[0];
  }
  else
  {
    return -1;
  }
}

else if (depth_value != "" && tip_angle_value == "0.0" && thru_flag == 0) // flat bottom
{
  printf("\nFLAT BOTTOM\n");

  int  array_dimension = 2;
  int  int_value[] = { 1, 1 };
  int  variable_type[] = { 301, 301 };
  char * hole_message = "Enter simple hole parameters";
  char menu_list[][16] = { "Diameter", "Depth" };
  char string_value[][31]  = { "", "" };
  double double_value[] = { 1.0, 1.0 };

  strcpy(string_value[0], diameter_value);
  strcpy(string_value[1], depth_value);

  retval = uc1613(   // show dialog-box
```

```
      hole_message,
      menu_list,
      array_dimension,
      int_value,
      double_value,
      string_value,
      variable_type);

    if (retval == 4)
    {
      diameter  = string_value[0];
      depth   = string_value[1];
    }
    else
    {
      return -1;
    }
  }

  else if (depth_value != "" && tip_angle_value != "0.0" && thru_flag == 0) // tip angle
  {
    printf("\nTIP ANGLE\n");

    int  array_dimension = 3;
    int  int_value[] = { 1, 1, 1 };
    int  variable_type[] = { 301, 301, 301  };
    char * hole_message = "Enter simple hole parameters";
    char menu_list[][16] = { "Diameter", "Depth", "Tip Angle" };
    char string_value[][31]  = { "", "", "" };
    double double_value[] = { 1.0, 1.0, 1.0 };

    strcpy(string_value[0], diameter_value);
    strcpy(string_value[1], depth_value);
    strcpy(string_value[2], tip_angle_value);

    retval = uc1613(   // show dialog-box
      hole_message,
      menu_list,
      array_dimension,
      int_value,
      double_value,
      string_value,
      variable_type);

    if (retval == 4)
    {
      diameter  = string_value[0];
      depth   = string_value[1];
      tip_angle = string_value[2];
    }
    else
    {
      return -1;
    }
  }

  UF_MODL_ask_feat_location(feature_id, location);
  UF_MODL_ask_feat_direction(feature_id, dir_z, dir_x);

  direction[0] = dir_z[0];
  direction[1] = dir_z[1];
  direction[2] = dir_z[2];

  UF_MODL_ask_feat_relatives(feature_id,       &num_parents,       &parent_array,       &num_children,
&children_array);
  UF_MODL_ask_feat_faces(parent_array[0], &block_faces);
  UF_MODL_ask_list_count(block_faces, &faces_count);

  for(int face=0; face<faces_count; face++)
  {
    UF_MODL_ask_list_item(block_faces, face, &block_face_id);
    get_face_norm(block_face_id, face_param, norm);

    if (dir_z[0] == (-1)*norm[0] && dir_z[1] == (-1)*norm[1] && dir_z[2] == (-1)*norm[2])
    {
      face_id = block_face_id;
    }

    if (dir_z[0] == norm[0] && dir_z[1] == norm[1] && dir_z[2] == norm[2] && thru_flag == 1)
    {
      face_thru = block_face_id;
    }
  }
```

```
  UF_MODL_create_list(&feat_list);
  UF_MODL_put_list_item(feat_list, feature_id);
  UF_MODL_delete_feature(feat_list);

  retval = UF_MODL_create_simple_hole(
           location,
           direction,
           diameter,
           depth,
           tip_angle,
           face_id,
           face_thru,
           &hole_id);

  feature_id = hole_id;

  return 0;
}



// USER DIALOG TO CREATE HOLE

int F_Hole_Simple::create_blind_flat_bottom()
{
  int   retval     = 0;
  double  location[3]  = {0.0, 0.0, 0.0};
  double  direction[3] = {0.0, 0.0, 0.0};
  double  face_param[2]= {0.0, 0.0};
  char *  dia_string   = "0.0000";
  char *  diameter    = dia_string;
  char *  depth;       // ignored if thru_hole
  char *  tip_angle;     // set to 0.0, so flat end if blind hole
  tag_t face_thru = NULL;  // face for thru face
  tag_t hole_id;     // id of the created simple hole
  tag_t sel_face_id;

  ////////////////////////////////////////////////////
  // prepare relative positioning
  UF_MODL_register_rpo_routine(UF_MODL_default_rpo_menu);

  ////////////////////////////////////////////////////
  // get location
  retval  =  select_location("Select  a  face  to  construct  the  hole",  sel_face_id,  location,
face_param);

  if (retval != 0)
  {
    return -1;
  }

  ////////////////////////////////////////////////////
  // get direction
  double normal[3] = {0.0, 0.0, 0.0};
  get_face_norm(sel_face_id, face_param, normal);
  UF_VEC3_negate(normal, direction);

  ////////////////////////////////////////////////////
  // get depth
  // get tip_angle
  // get diameter
  bool  thru_hole    = false;
  int   array_dimension  = 1;
  char  menu_list[][16]  = { "Depth"};
  int   int_value[]    = { 1 };           // maybe not used?
  double  double_value[]   = { 1.0};          // maybe not used?
  char  string_value[][31] = { "1.0000", "1.0000" };  // last item needed for diameter
  int   variable_type[]  = { 301 };           // string type used
      diameter     = string_value[1];

  ////////////////////////////////////////////////////
  // select diameter by selection list
  retval = get_hole_diameter(diameter);
  printf("\n\ndiameter  = %s", diameter);

  if (retval != 0)
  {
    return -1;
  }
  ////////////////////////////////////////////////////

  retval = uc1613(   // show dialog-box
```

```
    "Enter simple hole parameters",
    menu_list,
    array_dimension,
    int_value,
    double_value,
    string_value,
    variable_type
    );

  depth      = string_value[0]; // set depth
  tip_angle    = "0.0";     // set tip_angle


  if (retval != 4)
  {
    return -1; // no valid user input
  }
  // end of get diameter, depth, tip_angle
  ////////////////////////////////////////////////

  ////////////////////////////////////////////
  // now create the simple hole
  retval = UF_MODL_create_simple_hole(
    location,
    direction,
    diameter,
    depth,
    tip_angle,
    sel_face_id,
    face_thru,
    &hole_id);

  feature_id = hole_id;

  return 0;
}


// USER DIALOG TO CREATE HOLE

int F_Hole_Simple::create_blind_tip_angle()
{
  int   retval     = 0;
  double  location[3]  = {0.0, 0.0, 0.0};
  double  direction[3] = {0.0, 0.0, 0.0};
  double  face_param[2]= {0.0, 0.0};
  char *  dia_string   = "0.0000";
  char *  diameter    = dia_string;
  char *  depth;       // ignored if thru_hole
  char *  tip_angle;     // set to 0.0, so flat end if blind hole
  tag_t face_thru = NULL;  // face for thru face
  tag_t hole_id;     // id of the created simple hole
  tag_t sel_face_id;

  ////////////////////////////////////////////////////
  // prepare relative positioning
  UF_MODL_register_rpo_routine(UF_MODL_default_rpo_menu);

  ////////////////////////////////////////////////////
  // get location
  retval  =  select_location("Select  a  face  to  construct  the  hole",  sel_face_id,  location,
face_param);

  if (retval != 0)
  {
    return -1;
  }

  ////////////////////////////////////////////////////
  // get direction
  double normal[3] = {0.0, 0.0, 0.0};
  get_face_norm(sel_face_id, face_param, normal);
  UF_VEC3_negate(normal, direction);

  ////////////////////////////////////////////////////
  // get depth
  // get tip_angle
  // get diameter
  bool  thru_hole    = false;
  int   array_dimension  = 2;
  char  menu_list[][16]  = { "Depth",  "Tip angle"};
  int   int_value[]    = { 1,      1         };// maybe not used?
  double  double_value[]   = { 1.0,    1.0         };// maybe not used?
```

```
  char  string_value[][31] = { "1.0000", "1.0000", "1.0000"};// last item needed for diameter
  int   variable_type[]  = { 301 ,   301        };// string type used
      diameter     = string_value[2];

  ////////////////////////////////////////////////////
  // select diameter by selection list
  retval = get_hole_diameter(diameter);
  printf("\n\ndiameter  = %s", diameter);

  if (retval != 0)
  {
    return -1;
  }
  ////////////////////////////////////////////////////

  retval = uc1613(   // show dialog-box
    "Enter simple hole parameters",
    menu_list,
    array_dimension,
    int_value,
    double_value,
    string_value,
    variable_type
    );

  depth    = string_value[0];   // set depth
  tip_angle  = string_value[1];   // set tip_angle

  if (retval != 4)
  {
    return -1; // no valid user input
  }
  // end of get diameter, depth, tip_angle
  ////////////////////////////////////////////////////

  ////////////////////////////////////////////
  // now create the simple hole
  retval = UF_MODL_create_simple_hole(
    location,
    direction,
    diameter,
    depth,
    tip_angle,
    sel_face_id,
    face_thru,
    &hole_id);

  feature_id = hole_id;

  return 0;
}


// USER DIALOG TO CREATE HOLE

int F_Hole_Simple::create_blind_hole()
{
  int ip2=0;
  char cp3[][38] = {"Tip Angle", "Flat Bottom"};
  int ia6[3];
  int retval;
  int retval_1 = 0;


  retval_1 = uc1605("Select hole type", ip2, cp3, 2, ia6);


  if ((retval_1==1)||(retval_1==2)) // back or cancel
  {
    printf("\nERROR hole type\n");
    return -1;
  }

  else if (retval_1 == 3)         //ok
  {
    if (ia6[0] == 1 && ia6[1] == 0)
    {
      retval = create_blind_tip_angle();
    }

    if (ia6[0] == 0 && ia6[1] == 1)
    {
      retval = create_blind_flat_bottom();
```

```
    }

    if (ia6[0] == 0 && ia6[1] == 0)
    {
      retval = -1;
    }

  }
  return 0;
}


// USER DIALOG TO CREATE HOLE

int F_Hole_Simple::create_through_hole()
{
  int   retval     = 0;
  double  location[3]  = {0.0, 0.0, 0.0};
  double  direction[3] = {0.0, 0.0, 0.0};
  double  face_param[2]= {0.0, 0.0};
  char *  dia_string   = "0.0000";
  char *  diameter   = dia_string;
  char *  depth = 0;     // ignored if thru_hole
  char *  tip_angle = 0;   // set to 0.0, so flat end if blind hole
  tag_t face_thru;   // face for thru face
  tag_t hole_id;   // id of the created simple hole
  tag_t sel_face_id;

  ////////////////////////////////////////////////////
  // prepare relative positioning
  UF_MODL_register_rpo_routine(UF_MODL_default_rpo_menu);

  ////////////////////////////////////////////////////
  // get location
  retval  =  select_location("Select  a  face  to  construct  the  hole",  sel_face_id,  location,
face_param);

  if (retval != 0)
  {
    return -1;
  }

  ////////////////////////////////////////////////////
  // get direction
  double normal[3] = {0.0, 0.0, 0.0};
  get_face_norm(sel_face_id, face_param, normal);
  UF_VEC3_negate(normal, direction);

  ////////////////////////////////////////////////////
  // get diameter
    char  string_value[][31] = { "1.0000" };// last item needed for diameter
    diameter = string_value[0];

  ////////////////////////////////////////////////////
  // select diameter by selection list
  retval = get_hole_diameter(diameter);
  printf("\n\ndiameter  = %s", diameter);

  if (retval != 0)
  {
    return -1;
  }

  ////////////////////////////////////////////////////

  ////////////////////////////////////////////////////
  // get face_thru
//  select_face("Select Thru-Face for Festeval simple hole creation!", face_thru);

  double loc[3];
  double face_par[2];

  retval = select_location("Select the bottom face",face_thru, loc, face_par);

  if (retval != 0)
  {
    return -1;
  }
  ////////////////////////////////////////////
  // now create the simple hole
  retval = UF_MODL_create_simple_hole(
    location,
    direction,
```

```
    diameter,
    depth,
    tip_angle,
    sel_face_id,
    face_thru,
    &hole_id);

  feature_id = hole_id;

  return 0;
}

int F_Hole_Simple::set_face_names()
{
  printf("F_Hole_Simple::set_face_names\n");

  exists_virtual_face = 0;
  int index;
  int face_list_count = 0;
  int face_type;
  int flag_bottom_face = 0;
  int count_virtual = 1;
  char *face_name;
  char title[10] = "Face Type";
  tag_t Face;
  uf_list_p_t face_list;

  UF_ATTR_value_t value;
  value.type = UF_ATTR_string;
  value.value.string = "material face";

  UF_MODL_ask_feat_faces(feature_id, &face_list);
  UF_MODL_ask_list_count(face_list, &face_list_count);

  for(index=0; index<face_list_count; index++)
  {
    UF_MODL_ask_list_item(face_list, index, &Face);
    UF_MODL_ask_face_type(Face, &face_type);

    if(face_type == 16)      // CYLINDRICAL FACE
    {
      face_name ="CYLINDRICAL_FACE";
      UF_OBJ_set_name(Face, face_name);
      UF_ATTR_assign(Face, title, value);
    }

    else if(face_type == 22) // PLANAR FACE
    {
      face_name ="BOTTOM_FACE";
      UF_OBJ_set_name(Face, face_name);
      UF_ATTR_assign(Face, title, value);
      flag_bottom_face = 1;
    }

    else  if (face_type == 17) // CONICAL FACE
    {
      face_name ="BOTTOM_FACE";
      UF_OBJ_set_name(Face, face_name);
      UF_ATTR_assign(Face, title, value);
      flag_bottom_face = 1;
    }
  }

    if(flag_bottom_face == 0)
    {
      count_virtual = count_virtual + 1;
    }

  //  end of virtual faces identification
  if (count_virtual > 0)
  {
    exists_virtual_face = 1;
  }

  return 0;
}
```

K.2.13 F_Hole_CBore.cpp

```
// F_Hole_CBore

// counter bore hole feature

#include "F_Hole_CBore.h"
#include "F_Tools.h"


// CONSTRUCTOR (CREATE NEW FEATURE)

F_Hole_CBore::F_Hole_CBore()
{
  printf("F_Hole_CBore\n");
  this->init_feature();
}


// CONSTRUCTOR (FEATURE EXISTS IN UG)

F_Hole_CBore::F_Hole_CBore(tag_t feature_id)
{
  printf("F_Hole_CBore\n");
  this->init_feature(feature_id);
}


// CONSTRUCTOR (FEATURE EXISTS IN STEP)

F_Hole_CBore::F_Hole_CBore(CTransferObject step_param)
{
  printf("F_Hole_CBore\n");
  this->init_feature(step_param);
}


// TRANSFERS AND RETURNS PARAMETERS TO CREATE THE STEP INSTANCE

CTransferObject* F_Hole_CBore::get_step_param()
{
  int retval;
  CTransferObject *param;
  param = F_Feature::get_step_param(); // The feature-unspecific parameters

// (1)  Define all the feature-specific parameters
//    that need to be stored in the step model

  char*   diameter1;
  double    d1_upper, d1_lower;

  char*   diameter2;
  double    d2_upper, d2_lower;

  char*   depth1;
  double    dp1_upper, dp1_lower;

  char*   depth2;
  double    dp2_upper, dp2_lower;

  char*   tip_angle;
  double    ta_upper, ta_lower;

  int     thru_flag;

// (2)  Get all the parameters

  retval = UF_MODL_ask_c_bore_hole_parms( this->feature_id, 1,
                    &diameter1,
                    &diameter2,
                    &depth1,
                    &depth2,
                    &tip_angle,
                    &thru_flag);

  d1_upper = this->read_ug_attr_double("Hole Diameter Upper");
  d1_lower = this->read_ug_attr_double("Hole Diameter Lower");

  d2_upper = this->read_ug_attr_double("C-Bore Diameter Upper");
  d2_lower = this->read_ug_attr_double("C-Bore Diameter Lower");

  dp1_upper = this->read_ug_attr_double("Hole Depth Upper");
```

```
  dp1_lower = this->read_ug_attr_double("Hole Depth Lower");

  dp2_upper = this->read_ug_attr_double("C-Bore Depth Upper");
  dp2_lower = this->read_ug_attr_double("C-Bore Depth Lower");

  ta_upper = this->read_ug_attr_double("Tip Angle Upper");
  ta_lower = this->read_ug_attr_double("Tip Angle Lower");

//  (3) Transfer parameters to step parameters

  param->DoubleParam(F_TOOLS_str2dbl(diameter1), "Diameter1");
  param->DoubleParam(d1_upper, "Diameter1_Upper");
  param->DoubleParam(d1_lower, "Diameter1_Lower");

  param->DoubleParam(F_TOOLS_str2dbl(diameter2), "Diameter2");
  param->DoubleParam(d2_upper, "Diameter2_Upper");
  param->DoubleParam(d2_lower, "Diameter2_Lower");

  if (! thru_flag)
  {
    param->DoubleParam(F_TOOLS_str2dbl(depth1), "Depth1");
    param->DoubleParam(dp1_upper, "Depth1_Upper");
    param->DoubleParam(dp1_lower, "Depth1_Lower");
  }

  param->DoubleParam(F_TOOLS_str2dbl(depth2), "Depth2");
  param->DoubleParam(dp2_upper, "Depth2_Upper");
  param->DoubleParam(dp2_lower, "Depth2_Lower");

  param->DoubleParam(F_TOOLS_str2dbl(tip_angle), "TipAngle");
  param->DoubleParam(ta_upper, "TipAngle_Upper");
  param->DoubleParam(ta_lower, "TipAngle_Lower");

  param->IntParam(thru_flag, "ThroughBottom");

  return param;
}


// USER DIALOG TO EDIT FEATURE PARAMETERS

F_Hole_CBore::edit()
{
  int retval = 0;
  int edit = 1;
  int num_parents;
  int num_children;
  int faces_count = 0;
  int thru_flag = false;
  char * cbore_diameter;
  char * diameter;
  char * cbore_depth = NULL;
  char * depth = NULL;
  char * tip_angle = NULL;
  char * left;
  char * cbore_diameter_value;
  char * diameter_value;
  char * cbore_depth_value;
  char * depth_value;
  char * tip_angle_value;
  double direction[3];
  double location[3];
  double dir_x[3];
  double dir_z[3];
  double face_param[2];
  double norm[3];
  tag_t hole_id;
  tag_t exp_tag;
  tag_t face_id;
  tag_t * parent_array;
  tag_t * children_array;
  tag_t block_face_id;
  tag_t face_thru = NULL_TAG;
  uf_list_p_t feat_list;
  uf_list_p_t block_faces;

  UF_MODL_ask_c_bore_hole_parms(
              feature_id,
              edit,
              &cbore_diameter,
              &diameter,
              &cbore_depth,
              &depth,
```

```
            &tip_angle,
            &thru_flag);

UF_MODL_dissect_exp_string(cbore_diameter, &left, &cbore_diameter_value, &exp_tag);
UF_MODL_dissect_exp_string(diameter, &left, &diameter_value, &exp_tag);
UF_MODL_dissect_exp_string(cbore_depth, &left, &cbore_depth_value, &exp_tag);
UF_MODL_dissect_exp_string(depth, &left, &depth_value, &exp_tag);
UF_MODL_dissect_exp_string(tip_angle, &left, &tip_angle_value, &exp_tag);

if(thru_flag == 1) // thru hole
{
  int  array_dimension = 3;
  int  int_value[] = { 1, 1, 1};
  int  variable_type[] = { 301, 301, 301 };
  char * hole_message = "Enter hole parameters";
  char menu_list[][16] = { "C_Bore Diameter", "C_Bore Depth", "Diameter" };
  char string_value[][31] = { "", "", "" };
  double double_value[] = { 1.0, 1.0, 1.0 };

  strcpy(string_value[0], cbore_diameter_value);
  strcpy(string_value[1], cbore_depth_value);
  strcpy(string_value[2], diameter_value);

  retval = uc1613(   // show dialog-box
    hole_message,
    menu_list,
    array_dimension,
    int_value,
    double_value,
    string_value,
    variable_type);

  if (retval == 4)
  {
    cbore_diameter = string_value[0];
    cbore_depth    = string_value[1];
    diameter     = string_value[2];
  }
  else
  {
    return -1;
  }
}

else if (depth_value != "" && tip_angle_value == "0.0" && thru_flag == 0) // flat bottom
{
  int  array_dimension = 4;
  int  int_value[] = { 1, 1, 1, 1};
  int  variable_type[] = { 301, 301, 301, 301 };
  char * hole_message = "Enter hole parameters";
  char menu_list[][16] = { "C_Bore Diameter", "C_Bore Depth", "Diameter", "Depth" };
  char string_value[][31] = { "", "", "", "" };
  double double_value[] = { 1.0, 1.0, 1.0, 1.0 };

  strcpy(string_value[0], cbore_diameter_value);
  strcpy(string_value[1], cbore_depth_value);
  strcpy(string_value[2], diameter_value);
  strcpy(string_value[3], depth_value);

  retval = uc1613(   // show dialog-box
    hole_message,
    menu_list,
    array_dimension,
    int_value,
    double_value,
    string_value,
    variable_type);

  if (retval == 4)
  {
    cbore_diameter = string_value[0];
    cbore_depth    = string_value[1];
    diameter     = string_value[2];
    depth      = string_value[3];
  }
  else
  {
    return -1;
  }
}

else if (depth_value != "" && tip_angle_value != "0.0" && thru_flag == 0) // tip angle
{
```

```
    int  array_dimension = 5;
    int  int_value[] = { 1, 1, 1, 1, 1};
    int  variable_type[] = { 301, 301, 301, 301, 301 };
    char * hole_message = "Enter hole parameters";
    char menu_list[][16] = { "C_Bore Diameter", "C_Bore Depth", "Diameter", "Depth", "Tip Angle" };
    char string_value[][31]  = { "", "", "", "" };
    double double_value[] = { 1.0, 1.0, 1.0, 1.0 };

    strcpy(string_value[0], cbore_diameter_value);
    strcpy(string_value[1], cbore_depth_value);
    strcpy(string_value[2], diameter_value);
    strcpy(string_value[3], depth_value);
    strcpy(string_value[4], tip_angle_value);

    retval = uc1613(   // show dialog-box
      hole_message,
      menu_list,
      array_dimension,
      int_value,
      double_value,
      string_value,
      variable_type);

    if (retval == 4)
    {
      cbore_diameter = string_value[0];
      cbore_depth    = string_value[1];
      diameter    = string_value[2];
      depth     = string_value[3];
      tip_angle   = string_value[4];
    }
    else
    {
      return -1;
    }
  }


  UF_MODL_ask_feat_location(feature_id, location);
  UF_MODL_ask_feat_direction(feature_id, dir_z, dir_x);

  direction[0] = dir_z[0];
  direction[1] = dir_z[1];
  direction[2] = dir_z[2];

  UF_MODL_ask_feat_relatives(feature_id,       &num_parents,       &parent_array,       &num_children,
&children_array);
  UF_MODL_ask_feat_faces(parent_array[0], &block_faces);
  UF_MODL_ask_list_count(block_faces, &faces_count);

  for(int face=0; face<faces_count; face++)
  {
    UF_MODL_ask_list_item(block_faces, face, &block_face_id);
    get_face_norm(block_face_id, face_param, norm);

    if (dir_z[0] == (-1)*norm[0] && dir_z[1] == (-1)*norm[1] && dir_z[2] == (-1)*norm[2])
    {
      face_id = block_face_id;
    }

    if (dir_z[0] == norm[0] && dir_z[1] == norm[1] && dir_z[2] == norm[2] && thru_flag == 1)
    {
      face_thru = block_face_id;
    }

  }

  UF_MODL_create_list(&feat_list);
  UF_MODL_put_list_item(feat_list, feature_id);
  UF_MODL_delete_feature(feat_list);

  retval = UF_MODL_create_c_bore_hole(
           location,
           direction,
           diameter,
           depth,
           cbore_diameter,
           cbore_depth,
           tip_angle,
           face_id,
           face_thru,
           &hole_id);
```

```
  feature_id = hole_id;

  return 0;
}


// USER DIALOG TO CREATE HOLE

int F_Hole_CBore::create_blind_flat_bottom()
{
  int   retval     = 0;
  double  location[3]  = {0.0, 0.0, 0.0};
  double  direction[3] = {0.0, 0.0, 0.0};
  double  face_param[2]= {0.0, 0.0};
  char *  dia_string   = "0.0000";
  char *  diameter   = dia_string;
  char *  depth;       // ignored if thru_hole
  char *   c_bore_diameter;
  char *  c_bore_depth;
  char *  tip_angle;     // set to 0.0, so flat end if blind hole
  tag_t face_thru = NULL;  // face for thru face
  tag_t hole_id;     // id of the created simple hole
  tag_t sel_face_id;

  ////////////////////////////////////////////////////
  // prepare relative positioning
  UF_MODL_register_rpo_routine(UF_MODL_default_rpo_menu);

  ////////////////////////////////////////////////////
  // get location
  retval = select_location("Select face to create the hole", sel_face_id, location, face_param);

  if (retval != 0)
  {
    return -1;
  }

  ////////////////////////////////////////////////////
  // get direction
  double normal[3] = {0.0, 0.0, 0.0};
  get_face_norm(sel_face_id, face_param, normal);
  UF_VEC3_negate(normal, direction);
  char * hole_message = "Enter simple hole parameters";
  bool valid_input = false;

  ////////////////////////////////////////////////////
  // get depth
  // get tip_angle
  // get diameter
  bool  thru_hole    = false;
  int   array_dimension  = 3;
  char  menu_list[][16]  = { "Depth", "C_bore_diameter", "C_bore_depth" };
  int   int_value[]    = { 1,      1,      1         };// maybe not used?
  double  double_value[]   = { 1.0,    1.0,    1.0         };// maybe not used?
  char  string_value[][31] = { "1.0000", "1.0000", "1.0000", "1.0000"};// last item needed for diameter
  int   variable_type[]  = { 301 ,   301,    301         };// string type used
      diameter     = string_value[3];

  ////////////////////////////////////////////////////
  // select diameter by selection list
  retval = get_hole_diameter(diameter);
  printf("\n\ndiameter  = %s", diameter);
  ////////////////////////////////////////////////////

  do
  {
    retval = uc1613(         // show dialog-box
      hole_message,
      menu_list,
      array_dimension,
      int_value,
      double_value,
      string_value,
      variable_type
      );

    depth    = string_value[0];   // set depth
    c_bore_diameter= string_value[1];   // set c_bore_diameter
    c_bore_depth = string_value[2];   // set c_bore_depth
    tip_angle  = "0";          // set tip_angle


    switch (retval)
```

```
    {
      case 1: return -1;     break;  // back
      case 2: return -1;     break;  // cancel
      case 3:              // OK - no user input
      {
        hole_message = "You did not enter anything!!! Enter F_Hole_CBore parameters!!!";
        valid_input = false;
        break;
      }
      case 4: valid_input = true;  break;  // OK - user input
      case 8: return -2;     break;  // error, unable to bring up dialog
      default:break;
    }

  }

  while (!valid_input);


  // end of get diameter, depth, tip_angle
  ////////////////////////////////////////////////////

  ////////////////////////////////////////////
  // now create the counterbore hole

  retval = UF_MODL_create_c_bore_hole(
      location,
      direction,
      diameter,
      depth,
      c_bore_diameter,
      c_bore_depth,
      tip_angle,
      sel_face_id,
      face_thru,
      &hole_id);

  feature_id = hole_id;

  return 0;
}


// USER DIALOG TO CREATE HOLE

int F_Hole_CBore::create_blind_hole()
{
  int ip2=0;
  char cp3[][38] = {"Tip Angle", "Flat Bottom"};
  int ia6[3];
  int retval = 0;


  retval = uc1605("Select hole type", ip2, cp3, 2, ia6);


  if ((retval==1)||(retval==2)) // back or cancel
  {
    printf("\nERROR hole type\n");
    return -1;
  }

  else if (retval == 3)         //ok
  {
    if (ia6[0] == 1 && ia6[1] == 0)
    {
      retval = create_blind_tip_angle();
    }

    if (ia6[0] == 0 && ia6[1] == 1)
    {
      retval = create_blind_flat_bottom();
    }

    if (ia6[0] == 0 && ia6[1] == 0)
    {
      return -1;
    }
  }

  return 0;
}
```

```
// USER DIALOG TO CREATE HOLE

int F_Hole_CBore::create_blind_tip_angle()
{
  int   retval     = 0;
  double  location[3]  = {0.0, 0.0, 0.0};
  double  direction[3] = {0.0, 0.0, 0.0};
  double  face_param[2]= {0.0, 0.0};
  char *  dia_string   = "0.0000";
  char *  diameter   = dia_string;
  char *  depth       // ignored if thru_hole
  char *   c_bore_diameter;
  char *  c_bore_depth;
  char *  tip_angle;     // set to 0.0, so flat end if blind hole
  tag_t face_thru = NULL;  // face for thru face
  tag_t hole_id;     // id of the created simple hole
  tag_t sel_face_id;

  ////////////////////////////////////////////////////
  // prepare relative positioning
  UF_MODL_register_rpo_routine(UF_MODL_default_rpo_menu);

  ////////////////////////////////////////////////////
  // get location
  retval  =  select_location("Select  a  face  to  construct  the  hole",  sel_face_id,  location,
face_param);

  if (retval != 0)
  {
    return -1;
  }

  ////////////////////////////////////////////////////
  // get direction
  double normal[3] = {0.0, 0.0, 0.0};
  get_face_norm(sel_face_id, face_param, normal);
  UF_VEC3_negate(normal, direction);

  ////////////////////////////////////////////////////
  // get depth
  // get tip_angle
  // get diameter
  bool  thru_hole    = false;
  int   array_dimension  = 4;
  char  menu_list[][16]  = { "Depth", "C_bore_diameter", "C_bore_depth", "Tip angle"};
  int   int_value[]    = { 1,      1,      1,      1         };// maybe not used?
  double  double_value[]   = { 1.0,    1.0,    1.0,    1.0         };// maybe not used?
  char  string_value[][31] = { "1.0000", "1.0000", "1.0000", "1.0000", "1.0000"};//  last  item  needed
for diameter
  int   variable_type[]  = { 301 ,   301,    301,    301         };// string type used
      diameter     = string_value[4];

  ////////////////////////////////////////////////////
  // select diameter by selection list
  retval = get_hole_diameter(diameter);

  if (retval != 0)
  {
    return -1;
  }

  printf("\n\ndiameter  = %s", diameter);
  ////////////////////////////////////////////////////

  retval = uc1613(   // show dialog-box
    "Enter simple hole parameters",
    menu_list,
    array_dimension,
    int_value,
    double_value,
    string_value,
    variable_type
    );

  depth    = string_value[0];   // set depth
  c_bore_diameter= string_value[1];   // set c_bore_diameter
  c_bore_depth = string_value[2];   // set c_bore_depth
  tip_angle  = string_value[3];   // set tip_angle

  if (retval != 4)
  {
    return -1; // no valid user input
```

```
  }
  // end of get diameter, depth, tip_angle
  ////////////////////////////////////////////////

  ////////////////////////////////////////////
  // now create the counterbore hole

  retval = UF_MODL_create_c_bore_hole(
      location,
      direction,
      diameter,
      depth,
      c_bore_diameter,
      c_bore_depth,
      tip_angle,
      sel_face_id,
      face_thru,
      &hole_id);

  feature_id = hole_id;

  return 0;
}


// USER DIALOG TO CREATE HOLE

int F_Hole_CBore::create_through_hole()
{
  int   retval     = 0;
  double  location[3]  = {0.0, 0.0, 0.0};
  double  direction[3] = {0.0, 0.0, 0.0};
  double  face_param[2]= {0.0, 0.0};
  char *  dia_string   = "0.0000";
  char *  diameter   = dia_string;
  char *  depth    = 0;       // ignored if thru_hole
  char *   c_bore_diameter;
  char *  c_bore_depth;
  char *  tip_angle;     // set to 0.0, so flat end if blind hole
  tag_t face_thru = NULL;  // face for thru face
  tag_t hole_id;     // id of the created simple hole
  tag_t sel_face_id;

  ////////////////////////////////////////////////////
  // prepare relative positioning
  UF_MODL_register_rpo_routine(UF_MODL_default_rpo_menu);

  ////////////////////////////////////////////////////
  // get location
  retval  =  select_location("Select  a  face  to  construct  the  hole",  sel_face_id,  location,
face_param);

  if (retval != 0)
  {
    return -1;
  }

  ////////////////////////////////////////////////////
  // get direction
  double normal[3] = {0.0, 0.0, 0.0};
  get_face_norm(sel_face_id, face_param, normal);
  UF_VEC3_negate(normal, direction);

  ////////////////////////////////////////////////////
  // get depth
  // get tip_angle
  // get diameter
  bool  thru_hole    = true;
  int   array_dimension  = 2;
  char  menu_list[][16]  = { "C_bore_diameter", "C_bore_depth" };
  int   int_value[]    = { 1,      1          };// maybe not used?
  double  double_value[]   = { 1.0,    1.0          };// maybe not used?
  char  string_value[][31] = { "1.0000", "1.0000", "1.0000"  };// last item needed for diameter
  int   variable_type[]  = { 301 ,   301          };// string type used
      diameter     = string_value[2];

  ////////////////////////////////////////////////////
  // select diameter by selection list
  retval = get_hole_diameter(diameter);
  printf("\n\ndiameter  = %s", diameter);

  if (retval != 0)
  {
```

```
    return -1;
  }
  ////////////////////////////////////////////////////

  retval = uc1613(   // show dialog-box
    "Enter simple hole parameters",
    menu_list,
    array_dimension,
    int_value,
    double_value,
    string_value,
    variable_type
    );

  c_bore_diameter= string_value[0];   // set c_bore_diameter
  c_bore_depth = string_value[1];   // set c_bore_depth
  tip_angle  = "0";         // set tip_angle

  if (retval != 4)
  {
    return -1; // no valid user input
  }
  // end of get diameter, depth, tip_angle
  ////////////////////////////////////////////////////
  // get face_thru
  //  select_face("Select Thru-Face for Festeval simple hole creation!", face_thru);

  double loc[3];
  double face_par[2];
  select_location("Select the bottom face",face_thru, loc, face_par);
  ////////////////////////////////////////////
  // now create the counterbore hole

  retval = UF_MODL_create_c_bore_hole(
      location,
      direction,
      diameter,
      depth,
      c_bore_diameter,
      c_bore_depth,
      tip_angle,
      sel_face_id,
      face_thru,
      &hole_id);

  feature_id = hole_id;

  return 0;
}

int F_Hole_CBore::set_face_names()
{
  exists_virtual_face = 0;
  int index_1;
  int index_2;
  int face_list_count = 0;
  int face_type;
  int flag_bottom_face = 0;
  int count_virtual = 1;
  int count = 0;;
  int feat_count = 0;
  char *face_name;
  char title[10] = "Face Type";
  tag_t Face;
  tag_t edge;
  uf_list_p_t face_list;
  uf_list_p_t edge_list;
  uf_list_p_t features;

  UF_ATTR_value_t value;
  value.type = UF_ATTR_string;
  value.value.string = "material face";

  UF_MODL_ask_feat_faces(feature_id, &face_list);
  UF_MODL_ask_list_count(face_list, &face_list_count);

  for(index_1=0; index_1<face_list_count; index_1++)
  {
    UF_MODL_ask_list_item(face_list, index_1, &Face);
    UF_MODL_ask_face_type(Face, &face_type);

    if(face_type == 16)      // CYLINDRICAL FACE
    {
```

```
      UF_MODL_ask_face_edges(Face, &edge_list );
      UF_MODL_ask_list_count(edge_list, &count);

      for(index_2=0; index_2<count; index_2++)
      {
        UF_MODL_ask_list_item(edge_list, index_2, &edge);
        UF_MODL_ask_edge_feats(edge, &features);
        UF_MODL_ask_list_count(features, &feat_count);

        if(feat_count > 1)
        {
          face_name = "BORE_FACE";
          UF_OBJ_set_name(Face, face_name);
          UF_ATTR_assign(Face, title, value);
        }
        else if(feat_count == 1)
        {
          face_name ="CYLINDRICAL_FACE";
          UF_OBJ_set_name(Face, face_name);
          UF_ATTR_assign(Face, title, value);
        }
      }
    }

    else if(face_type == 22) // PLANAR FACE
    {
       UF_MODL_ask_face_edges(Face, &edge_list );
       UF_MODL_ask_list_count(edge_list, &count);

       if(count == 1)
       {
        face_name ="BOTTOM_FACE";
        flag_bottom_face = 1;
       }
       else if(count == 2)
       {
        face_name ="BORE_BOTTOM_FACE";
       }

       UF_OBJ_set_name(Face, face_name);
       UF_ATTR_assign(Face, title, value);
       flag_bottom_face = 1;
    }

    else  if (face_type == 17) // CONICAL FACE
    {
      face_name ="BOTTOM_FACE";
      UF_OBJ_set_name(Face, face_name);
      UF_ATTR_assign(Face, title, value);
      flag_bottom_face = 1;
    }
  }

    if(flag_bottom_face == 0)
    {
      count_virtual = count_virtual + 1;
    }

  //  end of virtual faces identification
  if (count_virtual > 0)
  {
    exists_virtual_face = 1;
  }

  return 0;
}
```

K.2.14 F_Hole_CSunk.cpp

```
// F_Hole_CSunk

// counter sunk hole feature

#include "F_Hole_CSunk.h"
#include "F_Tools.h"


// CONSTRUCTOR (CREATE NEW FEATURE)

F_Hole_CSunk::F_Hole_CSunk()
{
  printf("F_Hole_CSunk\n");
```

```
  this->init_feature();
}


// CONSTRUCTOR (FEATURE EXISTS IN UG)

F_Hole_CSunk::F_Hole_CSunk(tag_t feature_id)
{
  printf("F_Hole_CSunk\n");
  this->init_feature(feature_id);
}


// CONSTRUCTOR (FEATURE EXISTS IN STEP)

F_Hole_CSunk::F_Hole_CSunk(CTransferObject step_param)
{
  printf("F_Hole_CSunk\n");
  this->init_feature(step_param);
}


// TRANSFERS AND RETURNS PARAMETERS TO CREATE STEP INSTANCE

CTransferObject* F_Hole_CSunk::get_step_param()
{
  int retval;
  CTransferObject *param;
  param = F_Feature::get_step_param(); // The feature-unspecific parameters

// (1)  Define all the feature-specific parameters
//    that need to be stored in the step model

  char*   diameter1;
  double    d1_upper, d1_lower;

  char*   diameter2;
  double    d2_upper, d2_lower;

  char*   depth1;
  double    dp1_upper, dp1_lower;

  char*   csink_angle;
  double    ca_upper, ca_lower;

  char*   tip_angle;
  double    ta_upper, ta_lower;

  int     thru_flag;

// (2)  Get all the parameters

  retval = UF_MODL_ask_c_sunk_hole_parms( this->feature_id, 1,
                    &diameter1,
                    &diameter2,
                    &depth1,
                    &csink_angle,
                    &tip_angle,
                    &thru_flag);

  d1_upper = this->read_ug_attr_double("Hole Diameter Upper");
  d1_lower = this->read_ug_attr_double("Hole Diameter Lower");

  d2_upper = this->read_ug_attr_double("C-Sink Diameter Upper");
  d2_lower = this->read_ug_attr_double("C-Sink Diameter Lower");

  dp1_upper = this->read_ug_attr_double("Hole Depth Upper");
  dp1_lower = this->read_ug_attr_double("Hole Depth Lower");

  ca_upper = this->read_ug_attr_double("C-Sink Angle Upper");
  ca_lower = this->read_ug_attr_double("C-Sink Angle Lower");

  ta_upper = this->read_ug_attr_double("Tip Angle Upper");
  ta_lower = this->read_ug_attr_double("Tip Angle Lower");

//  (3) Transfer parameters to step parameters

  param->DoubleParam(F_TOOLS_str2dbl(diameter1), "Diameter1");
  param->DoubleParam(d1_upper, "Diameter1_Upper");
  param->DoubleParam(d1_lower, "Diameter1_Lower");

  param->DoubleParam(F_TOOLS_str2dbl(diameter2), "Diameter2");
  param->DoubleParam(d2_upper, "Diameter2_Upper");
```

```
  param->DoubleParam(d2_lower, "Diameter2_Lower");

  if (! thru_flag)
  {
    param->DoubleParam(F_TOOLS_str2dbl(depth1), "Depth1");
    param->DoubleParam(dp1_upper, "Depth1_Upper");
    param->DoubleParam(dp1_lower, "Depth1_Lower");
  }

  param->DoubleParam(F_TOOLS_str2dbl(csink_angle), "CsinkAngle");
  param->DoubleParam(ca_upper, "CsinkAngle_Upper");
  param->DoubleParam(ca_lower, "CsinkAngle_Lower");

  param->DoubleParam(F_TOOLS_str2dbl(tip_angle), "TipAngle");
  param->DoubleParam(ta_upper, "TipAngle_Upper");
  param->DoubleParam(ta_lower, "TipAngle_Lower");

  param->IntParam(thru_flag, "ThroughBottom");

  return param;
}


// USER DIALOG TO EDIT FEATURE PARAMETERS

F_Hole_CSunk::edit()
{
  int retval = 0;
  int edit = 1;
  int num_parents;
  int num_children;
  int faces_count = 0;
  int thru_flag = false;
  char * csunk_diameter;
  char * diameter;
  char * csunk_angle;
  char * depth = NULL;
  char * tip_angle = NULL;
  char * left;
  char * csunk_diameter_value;
  char * diameter_value;
  char * csunk_angle_value;
  char * depth_value;
  char * tip_angle_value;
  double direction[3];
  double location[3];
  double dir_x[3];
  double dir_z[3];
  double face_param[2];
  double norm[3];
  tag_t hole_id;
  tag_t exp_tag;
  tag_t face_id;
  tag_t * parent_array;
  tag_t * children_array;
  tag_t block_face_id;
  tag_t face_thru = NULL_TAG;
  uf_list_p_t feat_list;
  uf_list_p_t block_faces;

  UF_MODL_ask_c_sunk_hole_parms(
              feature_id,
              edit,
              &csunk_diameter,
              &diameter,
              &depth,
              &csunk_angle,
              &tip_angle,
              &thru_flag);


  UF_MODL_dissect_exp_string(csunk_diameter, &left, &csunk_diameter_value, &exp_tag);
  UF_MODL_dissect_exp_string(diameter, &left, &diameter_value, &exp_tag);
  UF_MODL_dissect_exp_string(depth, &left, &depth_value, &exp_tag);
  UF_MODL_dissect_exp_string(csunk_angle, &left, &csunk_angle_value, &exp_tag);
  UF_MODL_dissect_exp_string(tip_angle, &left, &tip_angle_value, &exp_tag);

  if(thru_flag == 1) // thru hole
  {
    printf("\nTHRU HOLE\n");

    int  array_dimension = 3;
    int  int_value[] = { 1, 1, 1};
```

```
  int  variable_type[] = { 301, 301, 301 };
  char * hole_message = "Enter hole parameters";
  char menu_list[][16] = { "C_Sunk Diameter", "C_Sunk Angle", "Diameter" };
  char string_value[][31] = { "", "", "" };
  double double_value[] = { 1.0, 1.0, 1.0 };

  strcpy(string_value[0], csunk_diameter_value);
  strcpy(string_value[1], csunk_angle_value);
  strcpy(string_value[2], diameter_value);

  retval = uc1613(   // show dialog-box
    hole_message,
    menu_list,
    array_dimension,
    int_value,
    double_value,
    string_value,
    variable_type);

  if (retval == 4)
  {
    csunk_diameter = string_value[0];
    csunk_angle    = string_value[1];
    diameter     = string_value[2];
  }
  else
  {
    return -1;
  }
}

else if (depth_value != "" && tip_angle_value == "0.0" && thru_flag == 0) // flat bottom
{
  printf("\nFLAT BOTTOM\n");

  int  array_dimension = 4;
  int  int_value[] = { 1, 1, 1, 1};
  int  variable_type[] = { 301, 301, 301, 301 };
  char * hole_message = "Enter hole parameters";
  char menu_list[][16] = { "C_Sunk Diameter", "C_Sunk Angle", "Diameter", "Depth" };
  char string_value[][31] = { "", "", "", "" };
  double double_value[] = { 1.0, 1.0, 1.0, 1.0 };

  strcpy(string_value[0], csunk_diameter_value);
  strcpy(string_value[1], csunk_angle_value);
  strcpy(string_value[2], diameter_value);
  strcpy(string_value[3], depth_value);

  retval = uc1613(   // show dialog-box
    hole_message,
    menu_list,
    array_dimension,
    int_value,
    double_value,
    string_value,
    variable_type);

  if (retval == 4)
  {
    csunk_diameter = string_value[0];
    csunk_angle    = string_value[1];
    diameter     = string_value[2];
    depth      = string_value[3];
  }
  else
  {
    return -1;
  }
}

else if (depth_value != "" && tip_angle_value != "0.0" && thru_flag == 0) // tip angle
{
  printf("\nTIP ANGLE\n");

  int  array_dimension = 5;
  int  int_value[] = { 1, 1, 1, 1, 1};
  int  variable_type[] = { 301, 301, 301, 301, 301 };
  char * hole_message = "Enter hole parameters";
  char menu_list[][16] = { "C_Sunk Diameter", "C_Sunk Angle", "Diameter", "Depth", "Tip Angle" };
  char string_value[][31] = { "", "", "", "" };
  double double_value[] = { 1.0, 1.0, 1.0, 1.0 };

  strcpy(string_value[0], csunk_diameter_value);
```

```
    strcpy(string_value[1], csunk_angle_value);
    strcpy(string_value[2], diameter_value);
    strcpy(string_value[3], depth_value);
    strcpy(string_value[4], tip_angle_value);

    retval = uc1613(   // show dialog-box
      hole_message,
      menu_list,
      array_dimension,
      int_value,
      double_value,
      string_value,
      variable_type);

    if (retval == 4)
    {
      csunk_diameter = string_value[0];
      csunk_angle    = string_value[1];
      diameter     = string_value[2];
      depth      = string_value[3];
      tip_angle    = string_value[4];
    }
    else
    {
      return -1;
    }
  }


  UF_MODL_ask_feat_location(feature_id, location);
  UF_MODL_ask_feat_direction(feature_id, dir_z, dir_x);

  direction[0] = dir_z[0];
  direction[1] = dir_z[1];
  direction[2] = dir_z[2];

  UF_MODL_ask_feat_relatives(feature_id,       &num_parents,       &parent_array,       &num_children,
&children_array);
  UF_MODL_ask_feat_faces(parent_array[0], &block_faces);
  UF_MODL_ask_list_count(block_faces, &faces_count);

  for(int face=0; face<faces_count; face++)
  {
    UF_MODL_ask_list_item(block_faces, face, &block_face_id);
    get_face_norm(block_face_id, face_param, norm);

    if (dir_z[0] == (-1)*norm[0] && dir_z[1] == (-1)*norm[1] && dir_z[2] == (-1)*norm[2])
    {
      face_id = block_face_id;
    }

    if (dir_z[0] == norm[0] && dir_z[1] == norm[1] && dir_z[2] == norm[2] && thru_flag == 1)
    {
      face_thru = block_face_id;
    }

  }

  UF_MODL_create_list(&feat_list);
  UF_MODL_put_list_item(feat_list, feature_id);
  UF_MODL_delete_feature(feat_list);


  retval = UF_MODL_create_c_sunk_hole(
             location,
             direction,
             diameter,
             depth,
             csunk_diameter,
             csunk_angle,
             tip_angle,
             face_id,
             face_thru,
             &hole_id);
  feature_id = hole_id;

  return 0;
}


// USER DIALOG TO CREATE HOLE

int F_Hole_CSunk::create_blind_flat_bottom()
```

```
{
  int   retval     = 0;
  double  location[3]  = {0.0, 0.0, 0.0};
  double  direction[3] = {0.0, 0.0, 0.0};
  double  face_param[2]= {0.0, 0.0};
  char *  dia_string   = "0.0000";
  char *  diameter    = dia_string;
  char *  depth;       // ignored if thru_hole
  char *   c_sunk_diameter;
  char *  c_sunk_angle;
  char *  tip_angle;     // set to 0.0, so flat end if blind hole
  tag_t face_thru = NULL;  // face for thru face
  tag_t hole_id;     // id of the created simple hole
  tag_t sel_face_id;

  ////////////////////////////////////////////////////
  // prepare relative positioning
  UF_MODL_register_rpo_routine(UF_MODL_default_rpo_menu);

  ////////////////////////////////////////////////////
  // get location
  retval  =  select_location("Select  a  face  to  construct  the  hole",  sel_face_id,  location,
face_param);

  if (retval != 0)
  {
    return -1;
  }

  ////////////////////////////////////////////////////
  // get direction
  double normal[3] = {0.0, 0.0, 0.0};
  get_face_norm(sel_face_id, face_param, normal);
  UF_VEC3_negate(normal, direction);

  ////////////////////////////////////////////////////
  // get depth
  // get tip_angle
  // get diameter
  bool  thru_hole    = false;
  int   array_dimension  = 3;
  char  menu_list[][16]  = { "Depth", "C_sunk_diameter", "C_sunk_angle" };
  int   int_value[]    = { 1,      1,      1         };// maybe not used?
  double  double_value[]   = { 1.0,    1.0,    1.0         };// maybe not used?
  char  string_value[][31] = { "1.0000", "1.0000", "1.0000", "1.0000"};// last item needed for diameter
  int   variable_type[]  = { 301 ,   301,    301         };// string type used
      diameter     = string_value[3];

  ////////////////////////////////////////////////////
  // select diameter by selection list
  retval = get_hole_diameter(diameter);
  printf("\n\ndiameter  = %s", diameter);

  if (retval != 0)
  {
    return -1;
  }
  ////////////////////////////////////////////////////

  retval = uc1613(   // show dialog-box
    "Enter simple hole parameters",
    menu_list,
    array_dimension,
    int_value,
    double_value,
    string_value,
    variable_type
    );

  depth     = string_value[0];   // set depth
  c_sunk_diameter= string_value[1];   // set c_bore_diameter
  c_sunk_angle = string_value[2];   // set c_bore_depth
  tip_angle   = "0.0";       // set tip_angle

  if (retval != 4)
  {
    return -1; // no valid user input
  }
  // end of get diameter, depth, tip_angle
  ////////////////////////////////////////////////////

  ////////////////////////////////////////////
  // now create the counterbore hole
```

```
  retval = UF_MODL_create_c_sunk_hole(
      location,
      direction,
      diameter,
      depth,
      c_sunk_diameter,
      c_sunk_angle,
      tip_angle,
      sel_face_id,
      face_thru,
      &hole_id);

  feature_id = hole_id;

  return 0;
}


// USER DIALOG TO CREATE HOLE

int F_Hole_CSunk::create_blind_hole()
{
  int ip2=0;
  char cp3[][38] = {"Tip Angle", "Flat Bottom"};
  int ia6[3];
  int retval;
  int retval_1 = 0;


  retval_1 = uc1605("Select hole type", ip2, cp3, 2, ia6);


  if ((retval_1==1)||(retval_1==2)) // back or cancel
  {
    printf("\nERROR hole type\n");
    return -1;
  }

  else if (retval_1 == 3)         //ok
  {
    if (ia6[0] == 1 && ia6[1] == 0)
    {
      retval = create_blind_tip_angle();
    }

    if (ia6[0] == 0 && ia6[1] == 1)
    {
      retval = create_blind_flat_bottom();
    }

    if (ia6[0] == 0 && ia6[1] == 0)
    {
      retval = -1;
    }
  }

  return 0;
}


// USER DIALOG TO CREATE HOLE

int F_Hole_CSunk::create_blind_tip_angle()
{
  int   retval     = 0;
  double  location[3]  = {0.0, 0.0, 0.0};
  double  direction[3] = {0.0, 0.0, 0.0};
  double  face_param[2]= {0.0, 0.0};
  char *  dia_string   = "0.0000";
  char *  diameter   = dia_string;
  char *  depth;       // ignored if thru_hole
  char *  c_sunk_diameter;
  char *  c_sunk_angle;
  char *  tip_angle;     // set to 0.0, so flat end if blind hole
  tag_t face_thru = NULL;  // face for thru face
  tag_t hole_id;     // id of the created simple hole
  tag_t sel_face_id;

  ////////////////////////////////////////////////////
  // prepare relative positioning
  UF_MODL_register_rpo_routine(UF_MODL_default_rpo_menu);
```

```
  ////////////////////////////////////////////////////
  // get location
  retval = select_location("Select a face to construct the hole", sel_face_id, location,
face_param);

  if (retval != 0)
  {
    return -1;
  }

  ////////////////////////////////////////////////////
  // get direction
  double normal[3] = {0.0, 0.0, 0.0};
  get_face_norm(sel_face_id, face_param, normal);
  UF_VEC3_negate(normal, direction);

  ////////////////////////////////////////////////////
  // get depth
  // get tip_angle
  // get diameter
  bool thru_hole    = false;
  int  array_dimension  = 4;
  char menu_list[][16]  = { "Depth", "C_sunk_diameter", "C_sunk_angle", "Tip angle"};
  int  int_value[]    = { 1,     1,     1,     1        };// maybe not used?
  double double_value[]   = { 1.0,   1.0,   1.0,   1.0        };// maybe not used?
  char string_value[][31] = { "1.0000", "1.0000", "1.0000", "1.0000", "1.0000"};//  last item needed
for diameter
  int  variable_type[]  = { 301 ,   301,   301,   301        };// string type used
      diameter     = string_value[4];

  ////////////////////////////////////////////////////
  // select diameter by selection list
  retval = get_hole_diameter(diameter);
  printf("\n\ndiameter  = %s", diameter);

  if (retval != 0)
  {
    return -1;
  }
  ////////////////////////////////////////////////////

  retval = uc1613(   // show dialog-box
    "Enter simple hole parameters",
    menu_list,
    array_dimension,
    int_value,
    double_value,
    string_value,
    variable_type
    );

  depth    = string_value[0];   // set depth
  c_sunk_diameter= string_value[1];   // set c_bore_diameter
  c_sunk_angle = string_value[2];   // set c_bore_depth
  tip_angle  = string_value[3];   // set tip_angle

  if (retval != 4)
  {
    return -1; // no valid user input
  }
  // end of get diameter, depth, tip_angle
  ////////////////////////////////////////////////////

  ////////////////////////////////////////////
  // now create the counterbore hole

  retval = UF_MODL_create_c_sunk_hole(
      location,
      direction,
      diameter,
      depth,
      c_sunk_diameter,
      c_sunk_angle,
      tip_angle,
      sel_face_id,
      face_thru,
      &hole_id);

  feature_id = hole_id;


  return 0;
}
```

```
// USER DIALOG TO CREATE HOLE

int F_Hole_CSunk::create_through_hole()
{
  int   retval     = 0;
  double  location[3]  = {0.0, 0.0, 0.0};
  double  direction[3] = {0.0, 0.0, 0.0};
  double  face_param[2]= {0.0, 0.0};
  char *  dia_string   = "0.0000";
  char *  diameter   = dia_string;
  char *  depth    = 0;       // ignored if thru_hole
  char *   c_sunk_diameter;
  char *  c_sunk_angle;
  char *  tip_angle;     // set to 0.0, so flat end if blind hole
  tag_t face_thru = NULL;  // face for thru face
  tag_t hole_id;     // id of the created simple hole
  tag_t sel_face_id;

  ////////////////////////////////////////////////////
  // prepare relative positioning
  UF_MODL_register_rpo_routine(UF_MODL_default_rpo_menu);

  ////////////////////////////////////////////////////
  // get location
  retval  =  select_location("Select  a  face  to  construct  the  hole",  sel_face_id,  location,
face_param);

  if (retval != 0)
  {
    return -1;
  }

  ////////////////////////////////////////////////////
  // get direction
  double normal[3] = {0.0, 0.0, 0.0};
  get_face_norm(sel_face_id, face_param, normal);
  UF_VEC3_negate(normal, direction);

  ////////////////////////////////////////////////////
  // get diameters
  bool  thru_hole    = true;
  int   array_dimension  = 2;
  char  menu_list[][16]  = { "C_sunk_diameter", "C_sunk_angle" };
  int   int_value[]    = { 1,      1          };// maybe not used?
  double  double_value[]   = { 1.0,    1.0,         };// maybe not used?
  char  string_value[][31] = { "1.0000", "1.0000", "1.0000"  };// last item needed for diameter
  int   variable_type[]  = { 301 ,   301          };// string type used
      diameter     = string_value[2];

  ////////////////////////////////////////////////////
  // select diameter by selection list
  retval = get_hole_diameter(diameter);
  printf("\n\ndiameter  = %s", diameter);

  if (retval != 0)
  {
    return -1;
  }
  ////////////////////////////////////////////////////

  retval = uc1613(   // show dialog-box
    "Enter simple hole parameters",
    menu_list,
    array_dimension,
    int_value,
    double_value,
    string_value,
    variable_type
    );

  c_sunk_diameter= string_value[0];   // set c_bore_diameter
  c_sunk_angle = string_value[1];   // set c_bore_depth
  tip_angle  = "0.0";       // set tip_angle

  if (retval != 4)
  {
    return -1; // no valid user input
  }
  // end of get diameters
  ////////////////////////////////////////////////////
```

```
  ////////////////////////////////////////////////
  // get face_thru
  //  select_face("Select Thru-Face for Festeval simple hole creation!", face_thru);

  double loc[3];
  double face_par[2];

  select_location("Select the bottom face",face_thru, loc, face_par);

  ////////////////////////////////////////////
  ////////////////////////////////////////////
  // now create the counterbore hole

  retval = UF_MODL_create_c_sunk_hole(
      location,
      direction,
      diameter,
      depth,
      c_sunk_diameter,
      c_sunk_angle,
      tip_angle,
      sel_face_id,
      face_thru,
      &hole_id);

  feature_id = hole_id;

  return 0;
}


int F_Hole_CSunk::set_face_names()
{
  printf("F_Hole_CSunk::set_face_names\n");

  exists_virtual_face = 0;
  int index;
  int face_list_count = 0;
  int face_type;
  int flag_bottom_face = 0;
  int count_virtual = 1;
  int edges_count = 0;
  char *face_name;
  char title[10] = "Face Type";
  tag_t Face;
  uf_list_p_t face_list;
  uf_list_p_t edges_list;

  UF_ATTR_value_t value;
  value.type = UF_ATTR_string;
  value.value.string = "material face";

  UF_MODL_ask_feat_faces(feature_id, &face_list);
  UF_MODL_ask_list_count(face_list, &face_list_count);

  for(index=0; index<face_list_count; index++)
  {
    UF_MODL_ask_list_item(face_list, index, &Face);
    UF_MODL_ask_face_type(Face, &face_type);

    if(face_type == 16)      // CYLINDRICAL FACE
    {
      face_name ="CYLINDRICAL_FACE";
      UF_OBJ_set_name(Face, face_name);
      UF_ATTR_assign(Face, title, value);
    }

    else if(face_type == 22) // PLANAR FACE
    {
      face_name ="BOTTOM_FACE";
      UF_OBJ_set_name(Face, face_name);
      UF_ATTR_assign(Face, title, value);
      flag_bottom_face = 1;
    }

    else  if (face_type == 17) // CONICAL FACE
    {
      UF_MODL_ask_face_edges(Face, &edges_list);
      UF_MODL_ask_list_count(edges_list, &edges_count);

      if(edges_count > 1)
      {
        face_name = "CSUNK_FACE";
```

```
        UF_OBJ_set_name(Face, face_name);
        UF_ATTR_assign(Face, title, value);
      }
      else if(edges_count == 1)
      {
        face_name ="BOTTOM_FACE";
        UF_OBJ_set_name(Face, face_name);
        UF_ATTR_assign(Face, title, value);
        flag_bottom_face = 1;
      }
    }

  }

    if(flag_bottom_face == 0)
    {
      count_virtual = count_virtual + 1;
    }

  //  end of virtual faces identification
  if (count_virtual > 0)
  {
    exists_virtual_face = 1;
  }

  return 0;
}
```

K.2.15 F_Planar_Face.cpp

```
// F_Planar_Face

// planar face feature

#include "F_Planar_Face.h"
#include "F_Tools.h"


// CONSTRUCTOR (CREATE NEW FEATURE)

F_Planar_Face::F_Planar_Face()
{
  printf("F_Planar_Face\n");

  this->init_feature();
}


// CONSTRUCTOR (FEATURE EXISTS IN UG)

F_Planar_Face::F_Planar_Face(tag_t feature_id)
{
  printf("F_Planar_Face\n");

  this->init_feature(feature_id);
}


// CONSTRUCTOR (FEATURE EXISTS IN STEP)

F_Planar_Face::F_Planar_Face(CTransferObject step_param)
{
  printf("F_Planar_Face\n");

  this->init_feature(step_param);
}


// TRANSFERS AND RETURNS PARAMETERS TO CREATE THE STEP INSTANCE

CTransferObject* F_Planar_Face::get_step_param()
{
  int retval;
  CTransferObject *param;
  param = F_Feature::get_step_param(); // The feature-unspecific parameters

// (1)  Define all the feature-specific parameters
//    that need to be stored in the step model

  int      num_objects;
  tag_t*     objects;
  UF_MODL_SWEEP_TRIM_object_p_t trim_ptr;
```

```
  char*    taper_angle;

  char*    limits[2];
  double     dp_upper, dp_lower;

  char*    offsets[2];
  double     region_point[3];
  unsigned charregion_specified;
  unsigned charsolid_creation;
  double     direction[3];

  double     uv_min_max[4];

// (2)  Get all the parameters

  retval = UF_MODL_ask_extrusion( this->feature_id,
                &num_objects,
                &objects,
                &trim_ptr,
                &taper_angle,
                limits,
                offsets,
                region_point,
                &region_specified,
                &solid_creation,
                direction);

  retval = UF_MODL_ask_face_uv_minmax(  objects[0],
                    uv_min_max);

  dp_upper = this->read_ug_attr_double("Limit 2 Upper");
  dp_lower = this->read_ug_attr_double("Limit 2 Lower");

//  (3) Transfer parameters to step parameters

  param->DoubleParam(uv_min_max[2], "Distance")
    ;
  param->DoubleArray(direction, "RemovalDirection", 3);

  param->DoubleParam(F_TOOLS_str2dbl(limits[1]), "Depth");
  param->DoubleParam(dp_upper, "Depth_Upper");
  param->DoubleParam(dp_lower, "Depth_Lower");

  return param;
}


// USER DIALOG TO EDIT FEATURE PARAMETERS

F_Planar_Face::edit()
{
  int num_objects;
  int retval = 0;
  char * taper_angle;
  char * limit[2];
  char * offsets[2];
  double point[3];
  double direction[3];
  tag_t * object;
  uf_list_p_t feat_list;
  uf_list_p_t object_list;
  uf_list_p_t planar_face_id;
  logical region_specified;
  logical solid_creation;
  UF_MODL_SWEEP_TRIM_object_p_t trim_ptr;
  UF_FEATURE_SIGN sign = UF_NEGATIVE;

  UF_MODL_ask_extrusion(
            feature_id,
            &num_objects,
            &object,
            &trim_ptr,
            &taper_angle,
            limit,
            offsets,
            point,
            &region_specified,
            &solid_creation,
            direction);

  UF_MODL_create_list(&object_list);

  for(int index=0; index<num_objects; index++)
```

```
  {
    UF_MODL_put_list_item(object_list, object[index]);
  }


  bool valid_input = false;
  int   array_dimension  = 1;
  int   int_value[]    = { 1 };
  int   variable_type[]  = { 301 };
  char * planar_face_message = "Enter planar face parameters";
  char  menu_list[][16]  = { "Depth" };
  char  string_value[][31] = { "" };
  double  double_value[]   = { 1.0 };

  strcpy(string_value[0], limit[1]);


  retval = uc1613(   // show dialog-box
      planar_face_message,
      menu_list,
      array_dimension,
      int_value,
      double_value,
      string_value,
      variable_type);

  if(retval == 4)
  {
    limit[1] = string_value[0];

    UF_MODL_create_list(&feat_list);
    UF_MODL_put_list_item(feat_list, feature_id);
    UF_MODL_delete_feature(feat_list);

    retval  =  UF_MODL_create_extruded(object_list,  taper_angle,  limit,  point,  direction,  sign,
&planar_face_id);
  }
  else
  {
    return -1;
  }

//  feature_id = planar_face_id;

  return 0;
}


// // USER DIALOG TO CREATE FEATURE

int F_Planar_Face::create_ug_feature()
{
  int retval =0;
  tag_t face_id;
  uf_list_p_t object;
  char *taper_angle;
  char *limit[2];
  double point[3];
  double direction[3];
  UF_FEATURE_SIGN sign = UF_NEGATIVE;
  uf_list_p_t planar_face_id;


  ////////////////////////////////////////////////
  // get location
  double  location[3];
  double face_param[2];
  retval = select_location("Select face to create the planar face", face_id, location, face_param);

  if (retval != 0)
  {
    return -1;
  }

  ////////////////////////////////////////////////
  // get direction
  double normal_cs[3]  = {0.0, 0.0, 0.0};
  get_face_norm(face_id, face_param, normal_cs);
  UF_VEC3_negate(normal_cs, direction);


  ////////////////////////////////////////
  // get planar_face limit
```

```
  bool valid_input = false;
  char * planar_face_message = "Enter planar_face limit";

  int   array_dimension  = 1;
  char  menu_list[][16]  = { "Depth" };
  int   int_value[]    = { 1   };  //maybe not used?
  double  double_value[]   = { 1.0   };  // maybe not used ?
  char  string_value[][31] = { "1.0000"};
  int   variable_type[]  = { 301   };  // string type used


  do
  {
    retval = uc1613(   // show dialog-box
      planar_face_message,
      menu_list,
      array_dimension,
      int_value,
      double_value,
      string_value,
      variable_type
      );

    switch (retval)
    {
      case 1: return -1;     break;  // back
      case 2: return -1;     break;  // cancel
      case 3:              // OK - no user input
      {
        planar_face_message = "You did not enter anything!!! Enter Planar Face parameters!!!";
        valid_input = false;
        break;
      }
      case 4: valid_input = true;  break;  // OK - user input
      case 8: return -2;     break;  // error, unable to bring up dialog
      default:break;
    }

  }

  while (!valid_input);


    limit[0] = "0.0";
    limit[1] = string_value[0]; // set pocket_size
    taper_angle = "0.0";

  // end of get planar_face limit
  ////////////////////////////////////////


    UF_MODL_create_list(&object);

    UF_MODL_put_list_item(object, face_id);


  retval   =   UF_MODL_create_extruded(object,   taper_angle,   limit,   point,   direction,   sign,
&planar_face_id);

//  feature_id = planar_face_id;

  return 0;
}


// VALIDATES FEATURE

bool F_Planar_Face::validate()
{
// ADD CODE HERE !!!
return true;
}


// SETS NAMES OF THE FACES

int F_Planar_Face::set_face_names()
{
// ADD CODE HERE !!!
return 0;
}

// STORES FACE INTERACTING FEATURES IN GLOBAL ATTRIBUTE faceint_features
```

```
uf_list_p_t F_Planar_Face::get_faceint_features()
{
// ADD CODE HERE !!!
return 0;
}
```

K.2.16 F_Pocket.cpp

```
// F_Pocket

// rectangular pocket feature

#include "F_Pocket.h"
#include "F_Tools.h"


// CONSTRUCTOR (CREATE NEW FEATURE)

F_Pocket::F_Pocket()
{
  printf("F_Pocket\n");

  this->init_feature();
}


// CONSTRUCTOR (FEATURE EXTSTS IN UG)

F_Pocket::F_Pocket(tag_t feature_id)
{
  printf("F_Pocket\n");

  this->init_feature(feature_id);
}


// CONSTRUCTOR (FEATURE EXISTS IN STEP)

F_Pocket::F_Pocket(CTransferObject step_param)
{
  printf("F_Pocket\n");

  this->init_feature(step_param);
}


// TRANSFERS AND RETURNS PARAMETERS TO CREATE THE STEP INSTANCE

CTransferObject* F_Pocket::get_step_param()
{
  int retval;
  CTransferObject *param;
  param = F_Feature::get_step_param(); // The feature-unspecific parameters

// (1)  Define all the feature-specific parameters
//    that need to be stored in the step model

  char*   length[3];
  double    l_upper, l_lower;
  double    w_upper, w_lower;
  double    d_upper, d_lower;

  char*   corner_rad;
  double    cr_upper, cr_lower;

  char*   floor_rad;
  double    fr_upper, fr_lower;

  char*   taper_angle;
  double    ta_upper, ta_lower;

// (2)  Get all the parameters

  retval = UF_MODL_ask_rect_pocket_parms( this->feature_id, 1,
                    length,
                    &corner_rad,
                    &floor_rad,
                    &taper_angle);

  l_upper = this->read_ug_attr_double("LengthX Upper");
  l_lower = this->read_ug_attr_double("LengthX Lower");
```

```
  w_upper = this->read_ug_attr_double("LengthY Upper");
  w_lower = this->read_ug_attr_double("LengthY Lower");

  d_upper = this->read_ug_attr_double("LengthZ Upper");
  d_lower = this->read_ug_attr_double("LengthZ Lower");

  cr_upper = this->read_ug_attr_double("Corner Radius Upper");
  cr_lower = this->read_ug_attr_double("Corner Radius Lower");

  fr_upper = this->read_ug_attr_double("Floor Radius Upper");
  fr_lower = this->read_ug_attr_double("Floor Radius Lower");

  ta_upper = this->read_ug_attr_double("Taper Angle Upper");
  ta_lower = this->read_ug_attr_double("Taper Angle Lower");

//  (3) Transfer parameters to step parameters

  param->DoubleParam(F_TOOLS_str2dbl(length[0]), "Length");
  param->DoubleParam(l_upper, "Length_Upper");
  param->DoubleParam(l_lower, "Length_Lower");

  param->DoubleParam(F_TOOLS_str2dbl(length[1]), "Width");
  param->DoubleParam(w_upper, "Width_Upper");
  param->DoubleParam(w_lower, "Width_Lower");

  param->DoubleParam(F_TOOLS_str2dbl(length[2]), "Depth");
  param->DoubleParam(d_upper, "Depth_Upper");
  param->DoubleParam(d_lower, "Depth_Lower");

  param->DoubleParam(F_TOOLS_str2dbl(corner_rad), "CornerRadius");
  param->DoubleParam(cr_upper, "CornerRadius_Upper");
  param->DoubleParam(cr_lower, "CornerRadius_Lower");

  param->DoubleParam(F_TOOLS_str2dbl(floor_rad), "BottomRadius");
  param->DoubleParam(fr_upper, "BottomRadius_Upper");
  param->DoubleParam(fr_lower, "BottomRadius_Lower");

  param->DoubleParam(F_TOOLS_str2dbl(taper_angle), "TaperAngle");
  param->DoubleParam(ta_upper, "TaperAngle_Upper");
  param->DoubleParam(ta_lower, "TaperAngle_Lower");

  return param;
}


// USER DIALOG TO EDIT FEATURE PARAMETERS

F_Pocket::edit()
{
  int retval = 0;
  int edit = 1;
  int num_parents;
  int num_children;
  int faces_count = 0;
  char * pocket_size[3];
  char * corner_radius ;
  char * floor_radius ;
  char * taper_angle = "0.0";
  char * left;
  char * right_0;
  char * right_1;
  char * right_2;
  char * right_3;
  char * right_4;
  double direction[3];
  double  x_direction[3];
  double location[3];
  double dir_y[3];
  double dir_x[3];
  double dir_z[3];
  double face_param[2];
  double norm[3];
  tag_t pocket_id;
  tag_t exp_tag;
  tag_t face_id;
  tag_t * parent_array;
  tag_t * children_array;
  tag_t block_face_id;
  uf_list_p_t feat_list;
  uf_list_p_t block_faces;
```

```
  UF_MODL_ask_rect_pocket_parms(feature_id,   edit,   pocket_size,   &corner_radius,   &floor_radius,
&taper_angle);

  UF_MODL_dissect_exp_string(pocket_size[0], &left, &right_0, &exp_tag);
  UF_MODL_dissect_exp_string(pocket_size[1], &left, &right_1, &exp_tag);
  UF_MODL_dissect_exp_string(pocket_size[2], &left, &right_2, &exp_tag);
  UF_MODL_dissect_exp_string(corner_radius, &left, &right_3, &exp_tag);
  UF_MODL_dissect_exp_string(floor_radius, &left, &right_4, &exp_tag);


  bool valid_input = false;
  int   array_dimension  = 5;
  int   int_value[]    = { 1, 1, 1, 1, 1 };
  int   variable_type[]  = { 301, 301, 301, 301, 301  };
  char * pocket_message = "Enter pocket parameters";
  char  menu_list[][16]  = { "X-Size", "Y-Size", "Depth", "Corner radius", "Floor radius" };
  char  string_value[][31] = { "", "", "", "", "" };
  double  double_value[]   = { 1.0, 1.0, 1.0, 1.0, 1.0 };

  strcpy(string_value[0], right_0);
  strcpy(string_value[1], right_1);
  strcpy(string_value[2], right_2);
  strcpy(string_value[3], right_3);
  strcpy(string_value[4], right_4);

  do
  {
    retval = uc1613(   // show dialog-box
      pocket_message,
      menu_list,
      array_dimension,
      int_value,
      double_value,
      string_value,
      variable_type);

    pocket_size[0] = string_value[0]; // set pocket_size
    pocket_size[1] = string_value[1]; // set pocket_size
    pocket_size[2] = string_value[2]; // set pocket_size
    corner_radius= string_value[3]; // set corner_radius
    floor_radius = string_value[4]; // set floor_radius
    taper_angle  = "0.0";     // set taper_angle


    // validation of corner_radius
    char *  stopstring;// only used for string to double conversion
    double  corner  = strtod( corner_radius, &stopstring ); // corner radius as a double

    switch (retval)
    {
      case 1: return -1;     break;  // back
      case 2: return -1;     break;  // cancel
      case 3:              // OK - no user input
      {
        pocket_message = "You did not enter anything!!! Enter Pocket parameters!!!";
        valid_input = false;
        break;
      }
      case 4:              // OK - user input
      {
        if (corner<0.5)
        {
          pocket_message = "No machine-tool available!!! Enter new corner radius!!!";
          valid_input = false;
        }
        else
        {
          valid_input = true;
        }
        break;
      }
      case 8: return -2;     break;  // error, unable to bring up dialog
      default:break;
    }

  }

  while (!valid_input);

  UF_MODL_ask_feat_location(feature_id, location);
  UF_MODL_ask_feat_direction(feature_id, dir_z, dir_x);

  direction[0] = dir_z[0];
```

```
  direction[1] = dir_z[1];
  direction[2] = dir_z[2];

  x_direction[0] = dir_x[0];
  x_direction[1] = dir_x[1];
  x_direction[2] = dir_x[2];

  UF_VEC3_cross(dir_z, dir_x, dir_y);

  UF_MODL_ask_feat_relatives(feature_id,       &num_parents,       &parent_array,       &num_children,
&children_array);
  UF_MODL_ask_feat_faces(parent_array[0], &block_faces);
  UF_MODL_ask_list_count(block_faces, &faces_count);

  for(int face=0; face<faces_count; face++)
  {
    UF_MODL_ask_list_item(block_faces, face, &block_face_id);
    get_face_norm(block_face_id, face_param, norm);

    if(dir_z[0] == (-1)*norm[0] && dir_z[1] == (-1)*norm[1] && dir_z[2] == (-1)*norm[2])
    {
      face_id = block_face_id;
    }
  }

  UF_MODL_create_list(&feat_list);
  UF_MODL_put_list_item(feat_list, feature_id);
  UF_MODL_delete_feature(feat_list);

  UF_MODL_create_rect_pocket(
             location,
             direction,
             x_direction,
             pocket_size,
             corner_radius,
             floor_radius,
             taper_angle,
             face_id,
             &pocket_id);

  feature_id = pocket_id;

  return 0;
}


// USER DIALOG TO CREATE NEW FEATURE

int F_Pocket::create_ug_feature()
{
  int retval;
  tag_t sel_face_id;
  double  location[3];
  double  face_param[3];
  double  direction[3];
  double  x_direction[3];
  char *  pocket_size[3];
  char *  corner_radius;
  char *  floor_radius;
  char *  taper_angle;
  tag_t pocket_id;
  tag_t edge_id;


  ////////////////////////////////////////////////
  // prepare relative positioning
  UF_MODL_register_rpo_routine(UF_MODL_default_rpo_menu);


  ////////////////////////////////////////////////
  // get location
  retval = select_location("Select face to create the pocket", sel_face_id, location, face_param);

  if (retval != 0)
  {
    return -1;
  }

  ////////////////////////////////////////////////
  // get direction
//  double normal_cs[3]  = {0.0, 0.0, 0.0};
  retval = get_face_norm(sel_face_id, face_param, normal_cs);
  if (retval != 0)
```

```
  {
    return -1;
  }

  UF_VEC3_negate(normal_cs, direction);

  ////////////////////////////////////////////////
  // get x_direction
  retval = select_direction("Select edge for X-direction of pocket", x_direction, edge_id);
  if (retval != 0)
  {
    return -1;
  }

  ////////////////////////////////////////
  // get pocket_size
  // get corner_radius
  // get taper_angle
  bool valid_input = false;
  char * pocket_message = "Enter pocket parameters";

  int   array_dimension  = 5;
  char  menu_list[][16]  = { "X-Size", "Y-Size", "Depth", "Corner radius", "Floor radius" };
  int   int_value[]    = { 1, 1, 1, 1, 1 };     //maybe not used?
  double  double_value[]   = { 1.0, 1.0, 1.0, 1.0, 1.0 }; // maybe not used ?
  char  string_value[][31] = { "1.0000", "1.0000", "1.0000", "1.0000", "1.0000"};// last item needed
for floor_radius
  int   variable_type[]  = { 301, 301, 301, 301, 301  };  // string type used

  do
  {
    retval = uc1613(   // show dialog-box
      pocket_message,
      menu_list,
      array_dimension,
      int_value,
      double_value,
      string_value,
      variable_type);

    pocket_size[0] = string_value[0]; // set pocket_size
    pocket_size[1] = string_value[1]; // set pocket_size
    pocket_size[2] = string_value[2]; // set pocket_size
    corner_radius= string_value[3]; // set corner_radius
    floor_radius = string_value[4]; // set floor_radius
    taper_angle  = "0.0";     // set taper_angle

    // validation of corner_radius
    char *  stopstring;// only used for string to double conversion
    double  corner  = strtod( corner_radius, &stopstring ); // corner radius as a double

    switch (retval)
    {
      case 1: return -1;     break;  // back
      case 2: return -1;     break;  // cancel
      case 3:              // OK - no user input
      {
        pocket_message = "You did not enter anything!!! Enter Pocket parameters!!!";
        valid_input = false;
        break;
      }
      case 4:              // OK - user input
      {
        if (corner<0.5)
        {
          pocket_message = "No machine-tool available!!! Enter new corner radius!!!";
          valid_input = false;
        }
        else
        {
          valid_input = true;
        }
        break;
      }
      case 8: return -2;     break;  // error, unable to bring up dialog
      default:break;
    }

  }

  while (!valid_input);
```

```
  ////////////////////////////////////////////////////////////////
  //  get floor_radius from list: 0.4, 0.8, 1.0
/*  retval = get_floor_radius(floor_radius);

  if (retval != 0)
  {
    return -1;
  }
*/
  ////////////////////////////////////////////
  // pocket create_ug_feature
  retval = UF_MODL_create_rect_pocket(
               location,
               direction,
               x_direction,
               pocket_size,
               corner_radius,
               floor_radius,
               taper_angle,
               sel_face_id,
               &pocket_id);

  feature_id = pocket_id;

  return 0;
}


// VALIDATES FEATURE

bool F_Pocket::validate()
{
  printf("F_Pocket::validate\n");

  // variable declarations
  int     retval;
  int     count;       // number of faces in the pocket
  int     cyl_face_count = 0;  // number of cylindrical faces in the pocket
  int     face_type;     // type of the face
  tag_t   body_id;     // id of the body which the feature belongs to
  uf_list_p_t face_list;     // list of faces in the feature
  uf_list_t *p_face;     // pointer to first element in face_list

  retval = UF_MODL_ask_feat_body (feature_id, &body_id); // get body of pocket
  retval = UF_MODL_ask_feat_faces(feature_id, &face_list);// get all faces of pocket
  retval = UF_MODL_ask_list_count(face_list, &count);    // get number of faces


  p_face = face_list;       // set pointer to first element in face_list

  while (p_face != NULL)
  {
    retval = UF_MODL_ask_face_type(p_face->eid, &face_type);

    if (face_type == 16)   // if cyl. face
    {
      cyl_face_count++;    // count the cyl. faces
    }

    if (face_type == 16 || face_type == 18) // if cyl. or sphere
    {
      tag_t offset_face;
      tag_t sheet_body_id;
      double  *rp2 = new double;   // offset distance
      double  *rp3 = new double;   // edge curve tolerance
      int   *lp4 = new int;    // not used
//      *rp2 = this->minimum_wall_size;   // set offset in mm
      *rp2 = 5.0;
      *rp3 = 0.00254;        // set tolerance in mm

      // prepare mark for UNDO after validation
      // ( UNDO the extract and offset process )
      UF_UNDO_user_visibility_t  pocket_validation_visibility= UF_UNDO_any_vis;
      UF_UNDO_mark_name_t      pocket_validation_mark_name = NULL;
      UF_UNDO_mark_id_t      pocket_validation_mark_id;
      int   number   =   UF_UNDO_set_mark(pocket_validation_visibility,   pocket_validation_mark_name,
&pocket_validation_mark_id);

      retval = UF_MODL_extract_face(p_face->eid, 0, &sheet_body_id);
      FTN(uf5450)(&sheet_body_id, rp2, rp3, lp4, &offset_face);

      int subtract = UF_MODL_operations (offset_face, body_id,  UF_NEGATIVE);
```

```
    //  UNDO the extract and offset process
      number = UF_UNDO_undo_to_mark(pocket_validation_mark_id, pocket_validation_mark_name);

      if (subtract == 0) // if 0, then offset_face is not in body => invalid
      {
        return false;// invalid Pocket
      }
    }

    p_face = p_face->next; // get pointer to next face in face_list
  }

  if (cyl_face_count < 5)    // less than 5 cyl. faces in body => invalid
  {
    return false;    // invalid Pocket
  }


  return true;       // valid

}


// SETS NAMES OF THE FACES

int F_Pocket::set_face_names()
{
  printf("F_Pocket::set_face_names\n");

  exists_virtual_face = false;
  int   color=0;
  int face_list_count = 0, retval;
  int count_cyl_face = 0;
  int  face_type = 0;
  int  flag_face_1=0, flag_face_2=0, flag_face_3=0, flag_face_4=0, flag_face_bottom=0;
  int count_material = 0;
  int edge_list_count = 0, cyl_edge_count = 0;
  int vertex_count = 0, edge_list_index = 0;
  int edge_type;
  int count_virtual = 1;
  double x_dir[3];
  double y_dir[3];
  double z_dir[3];
  double dir_x[3];
  double dir_y[3];
  double local_cs[3];
  double cross_product[3];
  double point1[3];
  double point2[3];
  double middle_point_1[3], middle_point_2[3], face_middle_point[3];
  double vec_diff[3];
  double face_point[3] = {0.0, 0.0, 0.0};
  double *face_point_p = face_point;
  double face_param[2];
  double angle;
  double angle_2;
  double grad_angle, grad_angle_2;
  double face_flat_point[2];
  double direction[3];
  double normal_cs[3];
  char *face_name;
  char title[10] = "Face Type";
  tag_t  csys_id = NULL;
  tag_t *exp_tags = NULL;
  tag_t Face;
  tag_t Edge;
  uf_list_p_t face_list;
  uf_list_p_t edge_list;
  uf_list_p_t flat_face_list = NULL;

  ////////////////////////////////////////////////////////////////////
  // ask the feature direction and location
  retval = get_direction_location(dir_x, dir_y, local_cs);

  ////////////////////////////////////////////////////////////////////
  // insert the coord system and ask the positioning parameters
  retval = create_coord_system(dir_x, dir_y, local_cs, cross_product, csys_id);

  x_dir[0] = cross_product[0];
  x_dir[1] = cross_product[1];
```

```
  x_dir[2] = cross_product[2];

  y_dir[0] = dir_x[0];
  y_dir[1] = dir_x[1];
  y_dir[2] = dir_x[2];

  UF_VEC3_cross(y_dir, x_dir, z_dir);

  UF_MODL_ask_feat_faces(feature_id, &face_list);
  UF_MODL_ask_list_count(face_list, &face_list_count);


  UF_ATTR_value_t value;
  value.type = UF_ATTR_string;
  value.value.string = "material face";

//  verify the normal_cs vector of the faces
  for (int face_list_index = 0; face_list_index < face_list_count; face_list_index++)
  {
    UF_MODL_ask_list_item(face_list, face_list_index, &Face);
    UF_MODL_ask_face_type(Face, &face_type);

    if (face_type == 16 )    // cylindrical face
    {
      retval = UF_MODL_ask_face_edges(Face, &edge_list);

    //  get number of edges in list
      retval = UF_MODL_ask_list_count(edge_list, &edge_list_count);

    //  loop through list of edges
      for (int edge_list_index = 0; edge_list_index < edge_list_count; edge_list_index++)
      {
      //  for each edge, verify to which face it belongs
        retval = UF_MODL_ask_list_item(edge_list, edge_list_index, &Edge);

        retval = UF_MODL_ask_edge_type(Edge, &edge_type);

        if (edge_type != UF_MODL_LINEAR_EDGE)
        {
          cyl_edge_count++;
          UF_MODL_ask_edge_verts(
                    Edge,
                    point1,
                    point2,
                    &vertex_count);

          if ( cyl_edge_count == 1 )
          {
            UF_VEC3_midpt( point1, point2, middle_point_1 );
          }

          if ( cyl_edge_count == 2 )
          {
            UF_VEC3_midpt(point1, point2, middle_point_2);
            UF_VEC3_midpt(middle_point_1, middle_point_2, face_middle_point);

            //  get_face_parm (Face, middle_point, face_param, face_point_p);
            UF_MODL_ask_face_parm(
                       Face,
                       face_middle_point,
                       face_param,
                       face_point);

          //  normal_cs vector to the corner faces
            UF_VEC3_sub(face_point, face_middle_point, vec_diff);

            UF_VEC3_angle_between(x_dir, vec_diff, z_dir, &angle);
            UF_VEC3_angle_between(z_dir, vec_diff, x_dir, &angle_2);
            grad_angle = (angle * 180.0) / (3.14159);
            grad_angle_2 = (angle_2 * 180.0) / (3.14159);

          //  present the conehead to the vector
            UF_DISP_conehead_attrb_s attrb;
            UF_DISP_conehead(UF_DISP_ALL_ACTIVE_VIEWS,
                       face_point, vec_diff, 0);
            UF_DISP_get_conehead_attrb(&attrb);
            attrb.color = 3;

          //  condition to avoid the code to find the floor faces

            if (grad_angle_2 >= 89 && grad_angle_2 <= 91 ||
              grad_angle_2 >= 269 && grad_angle_2 <= 271)
            {
```

```
          //  gambiarra, pois não conseguimos arrendondar o float

            if (grad_angle > 0 && grad_angle < 90)
            {
              face_name ="CORNER_2";
              UF_OBJ_set_name(Face, face_name);
              UF_ATTR_assign(Face, title, value);
            }

            if (grad_angle > 90 && grad_angle < 180)
            {
              face_name ="CORNER_3";
              UF_OBJ_set_name(Face, face_name);
              UF_ATTR_assign(Face, title, value);
            }

            if (grad_angle > 180 && grad_angle < 270)
            {
              face_name ="CORNER_4";
              UF_OBJ_set_name(Face, face_name);
              UF_ATTR_assign(Face, title, value);
            }

            if (grad_angle > 270 && grad_angle < 360)
            {
              face_name ="CORNER_1";
              UF_OBJ_set_name(Face, face_name);
              UF_ATTR_assign(Face, title, value);
            }
          }
            /* 20.12.99 calculates the distance between the normal_cs vector of the faces
             and the x_direction edge of the blank
            */
        }

      } // end of circular edge condition

    } // end of loop of edges of each face

  } // end of cylindrical face condition


  if (face_type == 22)   // planar face
  {
    UF_MODL_ask_face_parm(
              Face,
              face_middle_point,
              face_param,
              face_flat_point);

    get_face_norm(Face, face_param, direction);
    UF_VEC3_negate(direction, normal_cs);
    UF_MODL_put_list_item(flat_face_list, Face);

    if (y_dir[0] == normal_cs[0] && y_dir[1] == normal_cs[1]
      && y_dir[2] == normal_cs[2])
    {
      face_name ="FACE_1";
      UF_OBJ_set_name(Face, face_name);
      UF_ATTR_assign(Face, title, value);
      flag_face_1 = 1;
      count_material = count_material + 1;
    }

    if (x_dir[0] == normal_cs[0] && x_dir[1] == normal_cs[1]
      && x_dir[2] == normal_cs[2])
    {
      face_name ="FACE_2";
      UF_OBJ_set_name(Face, face_name);
      UF_ATTR_assign(Face, title, value);
      flag_face_2 = 1;
      count_material = count_material + 1;
    }

    if (y_dir[0] == (-1)*normal_cs[0] && y_dir[1] == (-1)*normal_cs[1]
      && y_dir[2] == (-1)*normal_cs[2])
    {
      face_name ="FACE_3";
      UF_OBJ_set_name(Face, face_name);
      UF_ATTR_assign(Face, title, value);
      flag_face_3 = 1;
      count_material = count_material + 1;      }
```

```
      if (x_dir[0] == (-1)*normal_cs[0] && x_dir[1] == (-1)*normal_cs[1]
        && x_dir[2] == (-1)*normal_cs[2])
      {
        face_name ="FACE_4";
        UF_OBJ_set_name(Face, face_name);
        UF_ATTR_assign(Face, title, value);
        flag_face_4 = 1;
        count_material = count_material + 1;
      }

      if (z_dir[0] == normal_cs[0] && z_dir[1] == normal_cs[1]
        && z_dir[2] == (-1)*normal_cs[2])
      {
        face_name = "FACE_BOTTOM";
        UF_OBJ_set_name(Face, face_name);
        UF_ATTR_assign(Face, title, value);
        flag_face_bottom = 1;
        count_material = count_material + 1;
      }


    //  present the conehead to the vector
      UF_DISP_conehead_attrb_s attrb;
      UF_DISP_get_conehead_attrb(&attrb);
      attrb.color = 3;
      UF_DISP_set_conehead_attrb(&attrb);
      UF_DISP_conehead(UF_DISP_ALL_ACTIVE_VIEWS,
                face_flat_point, normal_cs, 0);
    } //  end of flat face condition


  } //  end of loop of faces


//  virtual faces identifying
  value.value.string = "virtual face";

  if(flag_face_1 == 0)
  {
    UF_ATTR_assign(Face, title, value);
    count_virtual = count_virtual + 1;
  }

  if (flag_face_2 == 0)
  {
    UF_ATTR_assign(Face, title, value);
    count_virtual = count_virtual + 1;
  }

  if (flag_face_3 == 0)
  {
    UF_ATTR_assign(Face, title, value);
    count_virtual = count_virtual + 1;
  }

  if (flag_face_4 == 0)
  {
    UF_ATTR_assign(Face, title, value);
    count_virtual = count_virtual + 1;
  }

  if (flag_face_bottom == 0)
  {
    UF_ATTR_assign(Face, title, value);
    count_virtual = count_virtual + 1;
  }

  printf("\nCOUNT VIRTUAL:  %d\n", count_virtual);

//  end of virtual faces identification
  if (count_virtual > 1)
  {
    exists_virtual_face = true;
  }

  UF_DISP_display_rpo_dimensions(feature_id,
                -1,
                exp_tags,
                UF_DISP_VIEW_OF_LAST_CURSOR,
                UF_DISP_USE_ORIGINAL_COLOR, color);
  UF_DISP_refresh();

  return 0;
```

```
}


// STORES FACE INTERACTING FEATURES IN GLOBAL ATTRIBUTE faceint_features

uf_list_p_t F_Pocket::get_faceint_features()
{
  printf("F_Pocket::get_faceint_features\n");

  int guess1_given = 1;
  int guess2_given = 1;
  int vertex_count;
  int face_list_count = 0;
  int face_type = 0;
  double pt_on_ent1[3];
  double pt_on_ent2[3];
  double min_dist1;
  double min_dist2;
  double point1[3];
  double point2[3];
  double location[3];
  char face_name[7];
  bool exists_face_1 = 0;
  bool exists_face_2 = 0;
  bool exists_face_3 = 0;
  bool exists_face_4 = 0;
  bool exists_face_bottom = 0;
  tag_t edge1;
  tag_t edge2;
  tag_t object1 = NULL_TAG;
  tag_t object2 = NULL_TAG;
  tag_t first_face = NULL_TAG;
  tag_t second_face = NULL_TAG;
  tag_t Face = NULL_TAG;
  tag_t face_1 = NULL_TAG;
  tag_t face_2 = NULL_TAG;
  tag_t face_3 = NULL_TAG;
  tag_t face_4 = NULL_TAG;
  uf_list_p_t first_face_edges;
  uf_list_p_t second_face_edges;
  uf_list_p_t vertical_edges;
  uf_list_p_t face_list;
  uf_list_p_t virtedges_list;

  UF_MODL_create_list(&virtedges_list);

  UF_MODL_ask_feat_faces(feature_id, &face_list);
  UF_MODL_ask_list_count(face_list, &face_list_count);

  for (int face_list_index = 0; face_list_index < face_list_count; face_list_index++)
  {
    UF_MODL_ask_list_item(face_list, face_list_index, &Face);
    UF_MODL_ask_face_type(Face, &face_type);

    if (face_type == 22)
    {
      UF_OBJ_ask_name(Face, face_name);

      if(strcmp(face_name, "FACE_1") == 0)
      {
        exists_face_1 = 1;
        face_1 = Face;
      }
      else if(strcmp(face_name, "FACE_2") == 0)
      {
        exists_face_2 = 1;
        face_2 = Face;
      }
      else if(strcmp(face_name, "FACE_3") == 0)
      {
        exists_face_3 = 1;
        face_3 = Face;
      }
      else if(strcmp(face_name, "FACE_4") == 0)
      {
        exists_face_4 = 1;
        face_4 = Face;
      }
      else if(strcmp(face_name, "FACE_BOTTOM") == 0)
      {
        exists_face_bottom = 1;
      }
    }
```

```
}

if (exists_face_1 != 1)
{
  first_face = face_4;
  second_face = face_2;
}
else if (exists_face_2 != 1)
{
  first_face = face_1;
  second_face = face_3;
}
else if (exists_face_3 != 1)
{
  first_face = face_2;
  second_face = face_4;
}
else if (exists_face_4 != 1)
{
  first_face = face_3;
  second_face = face_1;
}
UF_MODL_ask_feat_location(feature_id, location);
//////////////////////////////////////////////////////
// get first face edges
UF_MODL_ask_face_edges(first_face, &first_face_edges);
vertical_edges = get_vertical_edges(first_face_edges);
//first edge
UF_MODL_ask_list_item(vertical_edges, 0, &edge1);
UF_MODL_ask_edge_verts(edge1, point1, point2, &vertex_count);

//distance between vertex (edge1) and central point (local_cs)
UF_MODL_ask_minimum_dist(object1, object2, guess1_given, location, guess2_given,
           point1, &min_dist1, pt_on_ent1, pt_on_ent2);

//second edge
UF_MODL_ask_list_item(vertical_edges, 1, &edge2);
UF_MODL_ask_edge_verts(edge2, point1, point2, &vertex_count);

//distance between vertex (edge2) and central point (local_cs)
UF_MODL_ask_minimum_dist(object1, object2, guess1_given, location, guess2_given,
            point1, &min_dist2, pt_on_ent1, pt_on_ent2);

if (min_dist1 < min_dist2)
{
  UF_MODL_put_list_item(virtedges_list, edge1);
}
else
{
  UF_MODL_put_list_item(virtedges_list, edge2);
}

//////////////////////////////////////////////////////
// get second face edges
UF_MODL_ask_face_edges(second_face, &second_face_edges);
vertical_edges = get_vertical_edges(second_face_edges);
//first edge
UF_MODL_ask_list_item(vertical_edges, 0, &edge1);
UF_MODL_ask_edge_verts(edge1, point1, point2, &vertex_count);
//distance between vertex (edge1) and central point (local_cs)
UF_MODL_ask_minimum_dist(object1, object2, guess1_given, location, guess2_given,
point2, &min_dist1, pt_on_ent1, pt_on_ent2);
//second edge
UF_MODL_ask_list_item(vertical_edges, 1, &edge2);
UF_MODL_ask_edge_verts(edge2, point1, point2, &vertex_count);

//distance between vertex (edge2) and central point (local_cs)
UF_MODL_ask_minimum_dist(object1, object2, guess1_given, location, guess2_given,
            point1, &min_dist2, pt_on_ent1, pt_on_ent2);

if (min_dist1 < min_dist2)
{
  UF_MODL_put_list_item(virtedges_list, edge1);
}
else
{
  UF_MODL_put_list_item(virtedges_list, edge2);
}

UF_MODL_create_list(&faceint_features);
faceint_features = get_edge_features(virtedges_list);

return faceint_features;
```

```
}


// GETS THE VERTICAL EDGES

uf_list_p_t F_Pocket::get_vertical_edges(uf_list_p_t face_edges)
{
  int count;
  uf_list_p_t vertical_edges;
  tag_t Edge;
  double point1[3];
  double point2[3];
  int vertex_count;
  double direction[3];


  UF_MODL_ask_list_count(face_edges, &count);
  UF_MODL_create_list(&vertical_edges);


  for (int index = 0; index < count; index++)
  {
    UF_MODL_ask_list_item(face_edges, index, &Edge);

    //get edge vertex
    UF_MODL_ask_edge_verts(Edge, point1, point2, &vertex_count);
//    printf("\n\n\n\nPOINT 1:  %f, %f, %f", point1[0], point1[1], point1[2]);
//    printf("\n\nPOINT 2:  %f, %f, %f", point2[0], point2[1], point2[2]);


    //get edge direction
    direction[0] = point2[0] - point1[0];
    direction[1] = point2[1] - point1[1];
    direction[2] = point2[2] - point1[2];
//    printf("\n\nDIRECTION:  %f, %f, %f", direction[0], direction[1], direction[2]);



    if (normal_cs[0] == 0 && normal_cs[1] == 0)// z direction
    {
      if(direction[0] == 0 && direction[1] == 0)  // z direction
      {
        UF_MODL_put_list_item(vertical_edges, Edge);
      }
    }

    if (normal_cs[0] == 0 && normal_cs[2] == 0)// y direction
    {
      if(direction[0] == 0 && direction[2] == 0)  // y direction
      {
        UF_MODL_put_list_item(vertical_edges, Edge);
      }
    }

    if (normal_cs[1] == 0 && normal_cs[2] == 0)// x direction
    {
      if(direction[1] == 0 && direction[2] == 0)  // x direction
      {
        UF_MODL_put_list_item(vertical_edges, Edge);
      }
    }

  }

  return vertical_edges;
}
```

K.2.17 F_Slot.cpp

```
// F_Slot

// rectangular slot feature

#include "F_Slot.h"
#include "F_Tools.h"


// CONSTRUCTOR (CREATE NEW FEATURE)

F_Slot::F_Slot()
{
  printf("F_Slot\n");
```

```
  this->init_feature();
}


// CONSTRUCTOR (FEATURE EXISTS IN UG)

F_Slot::F_Slot(tag_t feature_id)
{
  printf("F_Slot\n");

  this->init_feature(feature_id);
}


// CONSTRUCTOR (FEATURE EXISTS IN STEP)

F_Slot::F_Slot(CTransferObject step_param)
{
  printf("F_Slot\n");

  this->init_feature(step_param);
}


// TRANSFERS AND RETURNS PARAMETERS TO CREATE THE STEP INSTANCE

CTransferObject* F_Slot::get_step_param()
{
  int retval;
  CTransferObject *param;
  param = F_Feature::get_step_param(); // The feature-unspecific parameters

// (1)  Define all the feature-specific parameters
//    that need to be stored in the step model

  char* width;
  double  w_upper, w_lower;

  char* depth;
  double  dp_upper, dp_lower;

  char* distance;
  int   thru_flag;

// (2)  Get all the parameters


  retval = UF_MODL_ask_rect_slot_parms( this->feature_id, 1,
                    &width,
                    &depth,
                    &distance,
                    &thru_flag);

  w_upper = this->read_ug_attr_double("Width Upper");
  w_lower = this->read_ug_attr_double("Width Lower");

  dp_upper = this->read_ug_attr_double("Depth Upper");
  dp_lower = this->read_ug_attr_double("Depth Lower");

//  (3) Transfer parameters to step parameters

  param->DoubleParam(F_TOOLS_str2dbl(width), "Width");
  param->DoubleParam(w_upper, "Width_Upper");
  param->DoubleParam(w_lower, "Width_Lower");

  param->DoubleParam(F_TOOLS_str2dbl(depth), "Depth");
  param->DoubleParam(dp_upper, "Depth_Upper");
  param->DoubleParam(dp_lower, "Depth_Lower");

  param->DoubleParam(F_TOOLS_str2dbl(distance), "Distance");

  param->DoubleParam(0.0, "FirstRadius");
  param->DoubleParam(90.0, "FirstAngle");
  param->DoubleParam(0.0, "SecondRadius");
  param->DoubleParam(90.0, "SecondAngle");

  return param;
}


// USER DILOG TO EDIT FEATURE PARAMETERS
```

```
F_Slot::edit()
{
  printf("\nF_Slot::edit()\n");

  int retval = 0;
  int edit = 1;
  int thru_flag;
  int num_parents = 0;
  int num_children = 0;
  int faces_count = 0;
  char * width;
  char * depth;
  char * distance;
  char * exp;
  char * width_value;
  char * depth_value;
  double direction[3];
  double location[3];
  double  x_direction[3];
  double dir_y[3];
  double dir_x[3];
  double dir_z[3];
  double face_param[2];
  double norm[3];
  tag_t slot_id;
  tag_t exp_tag;
  tag_t face_id;
  tag_t face_thru_1;
  tag_t face_thru_2;
  tag_t * parent_array;
  tag_t * children_array;
  tag_t block_face_id;
  uf_list_p_t feat_list;
  uf_list_p_t block_faces;

  UF_MODL_ask_rect_slot_parms(this->feature_id, edit, &width, &depth, &distance, &thru_flag);
  UF_MODL_ask_thru_faces(this->feature_id, &face_thru_1, &face_thru_2);

  UF_MODL_dissect_exp_string(width, &exp, &width_value, &exp_tag);
  UF_MODL_dissect_exp_string(depth, &exp, &depth_value, &exp_tag);


  bool    valid_input = false;
  int   array_dimension  = 2;
  int   int_value[]    = { 1, 1 };
  int   variable_type[]  = { 301, 301 };
  char *  slot_message = "Enter slot parameters";
  char  menu_list[][16]  = { "Width", "Depth" };
  char  string_value[][31] = { "", "" };
  double  double_value[]   = { 1.0, 1.0 };

  strcpy(string_value[0], width_value);
  strcpy(string_value[1], depth_value);


    retval = uc1613(   // show dialog-box
      slot_message,
      menu_list,
      array_dimension,
      int_value,
      double_value,
      string_value,
      variable_type);

    width   = string_value[0]; // set width
    depth   = string_value[1]; // set depth


    ////////////////////////////////////////////////
    // validate of width
  do
  {
    char *  stopstring;// only used for string to double conversion
    double  d_width = strtod( width, &stopstring ); // width as a double

    switch (retval)
    {
      case 1: return -1;     break;  // back
      case 2: return -1;     break;  // cancel
      case 3:              // OK - no user input
      {
        slot_message = "You did not enter anything!!! Enter F_Slot parameters!!!";
        valid_input = false;
```

```
      break;
    }
    case 4:             // OK - user input
    {
      if (d_width<6)
      {
        slot_message = "No machine-tool available!!! Enter new slot parameters!!!";
        valid_input = false;
        strcpy(string_value[0], "6.0");
      }
      else
      {
        valid_input = true;
      }
      break;
    }
    case 8: return -2;     break;  // error, unable to bring up dialog
    default:break;
   }

  } while(!valid_input);


  UF_MODL_ask_feat_location(this->feature_id, location);
  UF_MODL_ask_feat_direction(this->feature_id, dir_y, dir_z);

  direction[0] = dir_z[0];
  direction[1] = dir_z[1];
  direction[2] = dir_z[2];

  x_direction[0] = dir_y[0];
  x_direction[1] = dir_y[1];
  x_direction[2] = dir_y[2];

  UF_VEC3_cross(dir_y, dir_z, dir_x);

  UF_MODL_ask_feat_relatives(this->feature_id,     &num_parents,     &parent_array,     &num_children,
&children_array);
  UF_MODL_ask_feat_faces(parent_array[0], &block_faces);
  UF_MODL_ask_list_count(block_faces, &faces_count);

  for(int face=0; face<faces_count; face++)
  {
    UF_MODL_ask_list_item(block_faces, face, &block_face_id);
    get_face_norm(block_face_id, face_param, norm);

    if(dir_y[0] == norm[0] && dir_y[1] == norm[1] && dir_y[2] == norm[2])
    {
      face_id = block_face_id;
    }
  }

  UF_MODL_create_list(&feat_list);
  UF_MODL_put_list_item(feat_list, this->feature_id);
  UF_MODL_delete_feature(feat_list);

  retval = UF_MODL_create_rect_slot(
            location,
            x_direction,
            direction,
            width,
            depth,
            distance, // not used
            face_id,
            face_thru_1,
            face_thru_2,
            &slot_id);

  this->feature_id = slot_id;

  return 0;
}


// USER DIALOG TO CREATE NEW FEATURE

int F_Slot::create_ug_feature()
{
  // declaration of variables
  int   retval = 0;
  double  face_param[2];
  double  location[3];   // location on target face for slot
  double  direction[3];  // direction of the slot
```

```
char *  width;       // width of slot
char *  depth;       // depth of slot
char *  distance = NULL; // not used, because thru-slot
tag_t sel_face_id;   // selected placement face
tag_t face_thru_1;   // start face for thru-slot
tag_t face_thru_2;   // end face for thru-slot
tag_t slot_id;     // ug-id for created slot
tag_t edge_id = NULL_TAG;

// prepare relative positioning
UF_MODL_register_rpo_routine(UF_MODL_default_rpo_menu);

//////////////////////////////////////////////////////
// get location
retval = select_location("Select face to create the slot", sel_face_id, location, face_param);

if (retval != 0)
{
  return -1;
}

//////////////////////////////////////////////////
// get tool_axis
retval = get_face_norm(sel_face_id, face_param, normal_cs);

//////////////////////////////////////////////////
// get direction
retval = select_direction("Select direction for Festeval slot creation!", direction, edge_id);

if (retval != 0)
{
  return -1;
}

//////////////////////////////////////////////////
// get width
// get depth
bool valid_input = false;
char * slot_message = "Enter slot parameters";

int   array_dimension  = 2;
char  menu_list[][16]  = { "Width",  "Depth" };
int   int_value[]    = { 6,      1   }; // maybe not used ?
double  double_value[]   = { 6.0,    1.0   }; // maybe not used ?
char  string_value[][31] = { "6.0000", "1.0000"};
int   variable_type[]  = { 301,    301   }; // string type used

do
{
    retval = uc1613(   // show dialog-box
    slot_message,
    menu_list,
    array_dimension,
    int_value,
    double_value,
    string_value,
    variable_type
    );

  width   = string_value[0]; // set width
  depth   = string_value[1]; // set depth


  //////////////////////////////////////////////////
  // validate of width
  char *  stopstring; // only used for string to double conversion
  double  d_width = strtod( width, &stopstring ); // width as a double

  switch (retval)
  {
    case 1: return -1;     break;  // back
    case 2: return -1;     break;  // cancel
    case 3:              // OK - no user input
    {
      slot_message = "You did not enter anything!!! Enter F_Slot parameters!!!";
      valid_input = false;
      break;
    }
    case 4:              // OK - user input
    {
      if (d_width<6)
      {
        slot_message = "No machine-tool available!!! Enter new slot parameters!!!";
```

```
        valid_input = false;
        strcpy(string_value[0], "6.0");
      }
      else
      {
        valid_input = true;
      }
      break;
    }
    case 8: return -2;     break;  // error, unable to bring up dialog
    default:break;
  }

} while(!valid_input);


////////////////////////////////////////////////
// get start and end face for thru slot
//  select_face("Select start face for Festeval thru slot creation!", face_thru_1);
//  select_face("Select end face for Festeval thru slot creation!", face_thru_2);

double loc[3];
double face_par[2];

retval = select_location("Select the first open face", face_thru_1, loc, face_par);
if (retval != 0)
{
  return -1;
}

retval = select_location("Select the second open face", face_thru_2, loc, face_par);
if (retval != 0)
{
  return -1;
}

////////////////////////////////////////
// now create the slot
retval = UF_MODL_create_rect_slot(
  location,
  normal_cs,
  direction,
  width,
  depth,
  distance, // not used
  sel_face_id,
  face_thru_1,
  face_thru_2,
  &slot_id);

  feature_id = slot_id;

return 0;
}


// VALIDATES THE FEATURE

bool F_Slot::validate()
{
  ////////////////////////////////////////////////
  // variable declarations
  int     retval;
  int     count;         // number of faces in the slot
  int     plane_face_count = 0;  // number of plane faces in the slot
  int     face_type;       // type of the face
  tag_t   body_id;       // id of the body which the feature belongs to
  uf_list_p_t face_list;       // list of faces in the feature
  uf_list_t *p_face;       // pointer to first element in face_list

  retval = UF_MODL_ask_feat_body (feature_id, &body_id); // get body of slot
  retval = UF_MODL_ask_feat_faces(feature_id, &face_list);// get all faces of slot
  retval = UF_MODL_ask_list_count(face_list, &count);    // get number of faces

  p_face = face_list;       // set pointer to first element in face_list

  while (p_face != NULL)
  {
    tag_t offset_face;
    tag_t sheet_body_id;
    double  *rp2 = new double;   // offset distance
    double  *rp3 = new double;   // edge curve tolerance
    int   *lp4 = new int;    // not used
```

```
//    *rp2 = this->minimum_wall_size;   // set offset in mm
    *rp2 = 5.0;
    *rp3 = 0.00254;        // set tolerance in mm

    ////////////////////////////////////////////////
    // prepare mark for UNDO after validate
    // ( UNDO the extract and offset process )
    UF_UNDO_user_visibility_t  slot_visibility= UF_UNDO_any_vis;
    UF_UNDO_mark_name_t      slot_mark_name = NULL;
    UF_UNDO_mark_id_t      slot_mark_id;
    int number = UF_UNDO_set_mark(slot_visibility, slot_mark_name, &slot_mark_id);

    retval = UF_MODL_extract_face(p_face->eid, 0, &sheet_body_id);
    FTN(uf5450)(&sheet_body_id, rp2, rp3, lp4, &offset_face);

    retval = UF_MODL_ask_face_type(p_face->eid, &face_type);

    bool valid = false;

    if (face_type == 16 )    //cyl. face
    {
      valid = (0 != UF_MODL_operations (offset_face, body_id,  UF_NEGATIVE));
    }
    else if (face_type == 22) //bounded plane
    {
      valid = (0 == UF_MODL_operations (offset_face, body_id,  UF_UNSIGNED));
      plane_face_count++;
    }
    else
    {
      cout << "\nError with face type in slot, type = " << face_type << endl;
    }

    ////////////////////////////////////////////////
    // UNDO the extract and offset process
    number = UF_UNDO_undo_to_mark(slot_mark_id, slot_mark_name);

    if (!valid)// if 0, then offset_face is not in body => invalid
    {
      return false;// invalid slot
    }

    p_face = p_face->next; // get pointer to next face in face_list
  }

  if (plane_face_count<3)
  {
    return false; // less than 3 faces => not valid
  }

  return true; // valid slot}
}


// SETS NAMES OF THE FACES

int F_Slot::set_face_names()
{
  printf("F_Slot::set_face_names\n");

  uf_list_p_t flat_face_list = NULL;
  tag_t face_1, face_2, face_3, face_4, face_bottom;


  //  get list with faces from feature
  //  local_cs variables

  int face_list_count = 0, retval;
  int count_cyl_face = 0;
  int face_type = 0;
  int flag_face_1 =0, flag_face_2=0, flag_face_3=0, flag_face_4=0, flag_face_bottom=0;
  int color=0;
  int count_material = 0;
  int edge_list_count = 0;
  int cyl_edge_count = 0;
  int vertex_count = 0;
  int edge_list_index = 0;
  int edge_type;
  int count_virtual = 1;
  char *face_name;
  double z_dir[3];
  double dir_x[3];
  double dir_y[3];
```

```
  double local_cs[3];
  double cross_product[3];
  double x_dir[3];
  double y_dir[3];
  double  point1[3];
  double  point2[3];
  double  middle_point_1[3], middle_point_2[3], face_middle_point[3];
  double  vec_diff[3];
  double face_point[3] = {0.0, 0.0, 0.0};
  double *face_point_p = face_point;
  double face_param[2];
  double angle;
  double angle_2;
  double grad_angle;
  double grad_angle_2;
//  double  face_param[2];
  double  face_flat_point[2];
//  double  face_middle_point[2];
  double  direction[3];
  double  normal_cs[3];
  tag_t  csys_id = NULL;
  tag_t *exp_tags = NULL;
  tag_t Face;
  tag_t Edge;
  uf_list_p_t face_list;
  uf_list_p_t edge_list;
  UF_DISP_conehead_attrb_s attrb;

  exists_virtual_face = false;

  ////////////////////////////////////////////////////////////////////
  // ask the feature direction and location
  retval = get_direction_location(dir_x, dir_y, local_cs);


  ////////////////////////////////////////////////////////////////////
  // insert the coord system and ask the positioning parameters
  retval = create_coord_system( dir_x, dir_y, local_cs, cross_product, csys_id);

  x_dir[0] = cross_product[0];
  x_dir[1] = cross_product[1];
  x_dir[2] = cross_product[2];

  y_dir[0] = dir_x[0];
  y_dir[1] = dir_x[1];
  y_dir[2] = dir_x[2];


  UF_VEC3_cross(y_dir, x_dir, z_dir);
  retval = UF_MODL_ask_feat_faces(feature_id, &face_list);
  retval = UF_MODL_ask_list_count(face_list, &face_list_count);


  ////////////////////////////////////////////////
  //  verify the normal_cs vector of the faces of the feature
  for (int face_list_index = 0; face_list_index < face_list_count; face_list_index++)
  {
    retval = UF_MODL_ask_list_item(face_list, face_list_index, &Face);
    retval = UF_MODL_ask_face_type(Face, &face_type);


    if (face_type == 16 )    //cylindrical face
    {
      retval = UF_MODL_ask_face_edges(Face, &edge_list);

    //  get number of edges in list
      retval = UF_MODL_ask_list_count(edge_list, &edge_list_count);

    //  loop through list of edges
      for (int edge_list_index = 0; edge_list_index < edge_list_count; edge_list_index++)
      {
      //  for each edge, verify to which face it belongs
        retval = UF_MODL_ask_list_item(edge_list, edge_list_index, &Edge);
        retval = UF_MODL_ask_edge_type(Edge, &edge_type);

        if (edge_type != UF_MODL_LINEAR_EDGE)
        {
          cyl_edge_count++;
          retval = UF_MODL_ask_edge_verts(
                     Edge,
                     point1,
                     point2,
                     &vertex_count);
```

```
if ( cyl_edge_count == 1 )
{
  UF_VEC3_midpt( point1, point2, middle_point_1 );
}

if ( cyl_edge_count == 2 )
{
  UF_VEC3_midpt(point1, point2, middle_point_2);
  UF_VEC3_midpt(middle_point_1, middle_point_2, face_middle_point);

//  get_face_parm (Face, middle_point, face_param, face_point_p);
  retval = UF_MODL_ask_face_parm(
             Face,
             face_middle_point,
             face_param,
             face_point);


//  normal_cs vector to the corner faces
  UF_VEC3_sub(face_point, face_middle_point, vec_diff);

  UF_VEC3_angle_between(x_dir, vec_diff, z_dir, &angle);
  UF_VEC3_angle_between(z_dir, vec_diff, x_dir, &angle_2);

  grad_angle = (angle * 180.0) / (3.14159);
  grad_angle_2 = (angle_2 * 180.0) / (3.14159);


//  present the conehead to the vector
  UF_DISP_conehead_attrb_s attrb;
  UF_DISP_conehead(UF_DISP_ALL_ACTIVE_VIEWS,
                face_point, vec_diff, 1);

  UF_DISP_get_conehead_attrb(&attrb);
  attrb.color = 3;

//  condition to avoid the code to find the floor faces
  if (grad_angle_2 >= 89 && grad_angle_2 <= 91 ||
    grad_angle_2 >= 269 && grad_angle_2 <= 271)
  {
  //  gambiarra, pois não conseguimos arrendondar o float

    if (grad_angle > 0 && grad_angle < 90)
    {
      face_name ="CORNER_2";
      UF_OBJ_set_name(Face, face_name);
    }

    if (grad_angle > 90 && grad_angle < 180)
    {
      face_name ="CORNER_3";
      UF_OBJ_set_name(Face, face_name);
    }

    if (grad_angle > 180 && grad_angle < 270)
    {
      face_name ="CORNER_4";
      UF_OBJ_set_name(Face, face_name);
    }

    if (grad_angle > 270 && grad_angle < 360)
    {
      face_name ="CORNER_1";
      UF_OBJ_set_name(Face, face_name);
    }

  }
    /* 20.12.99 calculates the distance between the normal_cs vector of the faces
     and the x_direction edge of the blank

    retval = UF_MODL_ask_edge_verts(
           sel_edge_id,
           x_direction_point1,
           x_direction_point2,
           &vertex_count);

    distance = sqrt( (pow((x_direction_point2[0] - face_point[0]), 2.0) +
             pow((x_direction_point2[1] - face_point[1]), 2.0) +
             pow((x_direction_point2[2] - face_point[2]), 2.0)));

    */
}
```

```
      } // end of circular edge condition

    } // end of loop of edges of each face

  } // end of cylindrical face condition

  if (face_type == 22)
  {
    retval = UF_MODL_ask_face_parm(
                  Face,
                  face_middle_point,
                  face_param,
                  face_flat_point);

    get_face_norm(Face, face_param, direction);
    UF_VEC3_negate(direction, normal_cs);


    if (y_dir[0] == normal_cs[0] && y_dir[1] == normal_cs[1]
      && y_dir[2] == normal_cs[2])
    {
      face_name ="FACE_1";
      UF_OBJ_set_name(Face, face_name);
      flag_face_1 = 1;
      count_material = count_material + 1;
      face_1 = Face;
    }


    if (x_dir[0] == normal_cs[0] && x_dir[1] == normal_cs[1]
      && x_dir[2] == normal_cs[2])
    {
      face_name ="FACE_2";
      UF_OBJ_set_name(Face, face_name);
      flag_face_2 = 1;
      count_material = count_material + 1;
      face_2 = Face;
    }


    if (y_dir[0] == (-1)*normal_cs[0] && y_dir[1] == (-1)*normal_cs[1]
      && y_dir[2] == (-1)*normal_cs[2])
    {
      face_name ="FACE_3";
      UF_OBJ_set_name(Face, face_name);
      flag_face_3 = 1;
      count_material = count_material + 1;
      face_3 = Face;
    }


    if (x_dir[0] == (-1)*normal_cs[0] && x_dir[1] == (-1)*normal_cs[1]
      && x_dir[2] == (-1)*normal_cs[2])
    {
      face_name ="FACE_4";
      UF_OBJ_set_name(Face, face_name);
      flag_face_4 = 1;
      count_material = count_material + 1;
      face_4 = Face;
    }

    if (z_dir[0] == normal_cs[0] && z_dir[1] == normal_cs[1]
      && z_dir[2] == normal_cs[2])
    {
      face_name ="FACE_BOTTOM";
      UF_OBJ_set_name(Face, face_name);
      flag_face_bottom = 1;
      count_material = count_material + 1;
      face_bottom = Face;
    }


  //  present the conehead to the vector
    UF_DISP_get_conehead_attrb(&attrb);
    attrb.color = 3;
    UF_DISP_set_conehead_attrb(&attrb);

    UF_DISP_conehead(UF_DISP_ALL_ACTIVE_VIEWS,
             face_flat_point, normal_cs, 1);
  } //  end of flat face condition
} //  end of loop of faces
```

```
//  virtual faces identifying

  if (flag_face_1 == 0)
  {
    count_virtual = count_virtual + 1;
  }

  if (flag_face_2 == 0)
  {
    count_virtual = count_virtual + 1;
  }

  if (flag_face_3 == 0)
  {
    count_virtual = count_virtual + 1;
  }

  if (flag_face_4 == 0)
  {
    count_virtual = count_virtual + 1;
  }

  if (flag_face_bottom == 0)
  {
    count_virtual = count_virtual + 1;
  }
//  end of virtual faces identification

  if (count_virtual > 1)
  {
    exists_virtual_face = true;
  }


  UF_DISP_display_rpo_dimensions(feature_id,
                  -1,
                exp_tags,
                UF_DISP_VIEW_OF_LAST_CURSOR,
                UF_DISP_USE_ORIGINAL_COLOR, color);
  UF_DISP_refresh();

  return 0;
}


// STORES FACE INTERACTING FEATURES IN GLOBAL ATTRIBUTE faceint_features

uf_list_p_t F_Slot::get_faceint_features()
{
  printf("F_Slot::get_faceint_features\n");

  int guess1_given = 1;
  int guess2_given = 1;
  int face_list_count = 0;
  int face_type = 0;
  int count = 0;
  double location[3];
  char face_name[7];
  bool exists_face_1 = false;
  bool exists_face_2 = false;
  bool exists_face_3 = false;
  bool exists_face_4 = false;
  bool exists_face_bottom = 0;
  tag_t object1 = NULL_TAG;
  tag_t object2 = NULL_TAG;
  tag_t first_face = NULL_TAG;
  tag_t second_face = NULL_TAG;
  tag_t Face = NULL_TAG;
  tag_t face_1 = NULL_TAG;
  tag_t face_2 = NULL_TAG;
  tag_t face_3 = NULL_TAG;
  tag_t face_4 = NULL_TAG;
  tag_t edge;
  uf_list_p_t first_face_edges;
  uf_list_p_t second_face_edges;
  uf_list_p_t vertical_edges;
  uf_list_p_t face_list;
  uf_list_p_t virtedges_list;

  UF_MODL_create_list(&virtedges_list);
```

```
UF_MODL_ask_feat_faces(feature_id, &face_list);
UF_MODL_ask_list_count(face_list, &face_list_count);
for (int face_list_index = 0; face_list_index < face_list_count; face_list_index++)
{
  UF_MODL_ask_list_item(face_list, face_list_index, &Face);
  UF_MODL_ask_face_type(Face, &face_type);
  if (face_type == 22)
  {
    UF_OBJ_ask_name(Face, face_name);

    if(strcmp(face_name, "FACE_1") == 0)
    {
      exists_face_1 = true;
      face_1 = Face;
    }
    else if(strcmp(face_name, "FACE_2") == 0)
    {
      exists_face_2 = true;
      face_2 = Face;
    }
    else if(strcmp(face_name, "FACE_3") == 0)
    {
      exists_face_3 = true;
      face_3 = Face;
    }
    else if(strcmp(face_name, "FACE_4") == 0)
    {
      exists_face_4 = true;
      face_4 = Face;
    }
    else if(strcmp(face_name, "FACE_BOTTOM") == 0)
    {
      exists_face_bottom = true;
    }
  }
}
if (exists_face_1 == false)
{
  first_face = face_4;
  second_face = face_2;
}
else if (exists_face_2 == false)
{
  first_face = face_1;
  second_face = face_3;
}
else if (exists_face_3 == false)
{
  first_face = face_2;
  second_face = face_4;
}
else if (exists_face_4 == false)
{
  first_face = face_3;
  second_face = face_1;
}

UF_MODL_ask_feat_location(feature_id, location);

UF_MODL_ask_face_edges(first_face, &first_face_edges);
vertical_edges = get_vertical_edges(first_face_edges);

for(int index=0; index<count;index++)
{
  UF_MODL_ask_list_item(vertical_edges, index, &edge);
  UF_MODL_put_list_item(virtedges_list, edge);
}

//////////////////////////////////////////////////////
// get second face edges
UF_MODL_ask_face_edges(second_face, &second_face_edges);
vertical_edges = get_vertical_edges(second_face_edges);

UF_MODL_ask_list_count(vertical_edges, &count);
printf("\nCOUNT:  %d\n", count);

for(index=0; index<count;index++)
{
  UF_MODL_ask_list_item(vertical_edges, index, &edge);
  UF_MODL_put_list_item(virtedges_list, edge);
}
```

```
  UF_MODL_create_list(&faceint_features);
  faceint_features = get_edge_features(virtedges_list);

  return faceint_features;
}

uf_list_p_t F_Slot::get_vertical_edges(uf_list_p_t face_edges)
{
  int count;
  int vertex_count;
  double point1[3];
  double point2[3];
  double direction[3];
  tag_t Edge;
  uf_list_p_t vertical_edges;

  UF_MODL_ask_list_count(face_edges, &count);
  UF_MODL_create_list(&vertical_edges);

  for (int index = 0; index < count; index++)
  {
    UF_MODL_ask_list_item(face_edges, index, &Edge);

    //get edge vertex
    UF_MODL_ask_edge_verts(Edge, point1, point2, &vertex_count);

    //get edge direction
    direction[0] = point2[0] - point1[0];
    direction[1] = point2[1] - point1[1];
    direction[2] = point2[2] - point1[2];
    printf("\nNORMAL_CS:  %f, %f, %f\n", normal_cs[0], normal_cs[1], normal_cs[2]);
    printf("\nDIRECTION:  %f, %f, %f\n", direction[0], direction[1], direction[2]);

    if (normal_cs[0] == 0 && normal_cs[1] == 0)// z direction
    {
      if(direction[0] == 0 && direction[1] == 0)  // z direction
      {
        UF_MODL_put_list_item(vertical_edges, Edge);
      }
    }

    if (normal_cs[0] == 0 && normal_cs[2] == 0)// y direction
    {
      if(direction[0] == 0 && direction[2] == 0)  // y direction
      {
        UF_MODL_put_list_item(vertical_edges, Edge);
      }
    }

    if (normal_cs[1] == 0 && normal_cs[2] == 0)// x direction
    {
      if(direction[1] == 0 && direction[2] == 0)  // x direction
      {
        UF_MODL_put_list_item(vertical_edges, Edge);
      }
    }
  }

  ////////////////////////////////////////////////
  UF_MODL_ask_list_count(vertical_edges, &count);
    printf("\nCOUNT VERTICAL EDGES:  %d\n", count);
  ////////////////////////////////////////////////


  return vertical_edges;
}
```

K.2.18 F_Thread.cpp

```
// F_Thread

// symbolic thread feature

#include "F_Thread.h"
#include "F_Tools.h"


// CONSTRUCTOR (CREATE NEW FEATURE)

F_Thread::F_Thread()
{
  printf("F_Thread\n");
```

```
  this->init_feature();
}


// CONSTRUCTOR (FEATURE EXISTS IN UG)

F_Thread::F_Thread(tag_t feature_id)
{
  printf("F_Thread\n");

  this->init_feature(feature_id);
}


// CONSTRUCTOR (FEATURE EXISTS IN STEP)

F_Thread::F_Thread(CTransferObject step_param)
{
  printf("F_Thread\n");

  this->init_feature(step_param);
}


// TRANSFERS AND RETURNS PARAMETERS TO CREATE THE STEP INSTANCE

CTransferObject* F_Thread::get_step_param()
{
  int retval;
  CTransferObject *param;
  param = F_Feature::get_step_param(); // The feature-unspecific parameters

// (1)  Define all the feature-specific parameters
//    that need to be stored in the step model

  UF_MODL_symb_thread_data_t parameters;

  double  l_upper, l_lower;
  double  mjd_upper, mjd_lower;
  double  mnd_upper, mnd_lower;
  double  pt_upper, pt_lower;

// (2)  Get all the parameters


  retval = UF_MODL_ask_symb_thread_parms( this->feature_id,
                    &parameters);

  l_upper = this->read_ug_attr_double("Length Upper");
  l_lower = this->read_ug_attr_double("Length Lower");

  mjd_upper = this->read_ug_attr_double("Major Diameter Upper");
  mjd_lower = this->read_ug_attr_double("Major Diameter Lower");

  mnd_upper = this->read_ug_attr_double("Minor Diameter Upper");
  mnd_lower = this->read_ug_attr_double("Minor Diameter Lower");

  pt_upper = this->read_ug_attr_double("Pitch Upper");
  pt_lower = this->read_ug_attr_double("Pitch Lower");

// (3) Transfer parameters to step parameters

  param->DoubleParam(F_TOOLS_str2dbl(parameters.length), "Length");
  param->DoubleParam(l_upper, "Length_Upper");
  param->DoubleParam(l_lower, "Length_Lower");

  param->DoubleParam(F_TOOLS_str2dbl(parameters.major_dia), "Major_Dia");
  param->DoubleParam(mjd_upper, "Major_Dia_Upper");
  param->DoubleParam(mjd_lower, "Major_Dia_Lower");

  param->DoubleParam(F_TOOLS_str2dbl(parameters.minor_dia), "Minor_Dia");
  param->DoubleParam(mnd_upper, "Minor_Dia_Upper");
  param->DoubleParam(mnd_lower, "Minor_Dia_Lower");

  param->DoubleParam(F_TOOLS_str2dbl(parameters.pitch), "Pitch");
  param->DoubleParam(pt_upper, "Pitch_Upper");
  param->DoubleParam(pt_lower, "Pitch_Lower");

  param->Strings(parameters.form, "Form");
  param->Strings(parameters.callout, "Callout");
  param->Strings(parameters.method, "Form");
```

```
  param->DoubleArray(parameters.axis_direction, "Dir_x", 3);

  return param;
}


// USER DIALOG TO EDIT FEATURE PARAMETERS

F_Thread::edit()
{
// ADD CODE HERE !!!
uc1601("ERROR : Function not implemented for F_Thread yet.", 1);
}


// USER DIALOG TO CREATE NEW FEATURE

int F_Thread::create_ug_feature()
{
  int retval = 0;
  int ip2=0;
  int instance[2];
  int rotation[2];
  int length[2];
  int taper[2];
  char thread_type[][38] = {"Instances", "No Instances"};
  char thread_rotation[][38] = {"Right Hand", "Left Hand"};
  char thread_length_flag[][38] = {"Full Thread", "Fixed Thread"};
  char thread_tapered[][38] = {"Non Tapered", "Tapered"};
  double face_param[2];
  double norm[3];
  double direction[3];
  double loc[3];
  double face_par[2];
  tag_t thread_id;
  tag_t cyl_face;
  tag_t start_face;
  UF_MODL_symb_thread_data_t thread;

  retval = select_cylindrical_face("Select a cylindrical face to make the thread", cyl_face);

  if (retval != 0)   // back or cancel....
  {
    return -1;
  }

  thread.cyl_face = cyl_face;

  retval = select_location("Select start face to create the thread", start_face, loc, face_par);

  if (retval != 0)
  {
    return -1;
  }

  thread.start_face = start_face;

  get_face_norm(start_face, face_param, norm);
  UF_VEC3_negate(norm, direction);

  thread.axis_direction[0] = direction [0];
  thread.axis_direction[1] = direction [1];
  thread.axis_direction[2] = direction [2];

  ////////////////////////////////////////////////////////////////////////////
  // thread instances

  retval = uc1605("Select thread type", ip2, thread_type, 2, instance);

  if ((retval==1)||(retval==2)) // back or cancel
  {
    printf ("\nERROR hole type\n");
    return -1;
  }

  else if (retval == 3)    //ok
  {
    if (instance[0] == 1 && instance[1] == 0)
    {
      thread.include_instances = UF_MODL_INCL_INSTANCES;
    }
    if (instance[0] == 0 && instance[1] == 1)
    {
```

```
    thread.include_instances = UF_MODL_NO_INSTANCES;
  }
}

////////////////////////////////////////////////////////////////////////////
// thread rotation

retval = uc1605("Select thread rotation", ip2, thread_rotation, 2, rotation);

if ((retval==1)||(retval==2)) // back or cancel
{
  printf("\nERROR hole type\n");
  return -1;
}

else if (retval == 3)   //ok
{
  if (rotation[0] == 1 && rotation[1] == 0)
  {
    thread.rotation = UF_MODL_RIGHT_HAND;
  }
  if (rotation[0] == 0 && rotation[1] == 1)
  {
    thread.rotation = UF_MODL_LEFT_HAND;
  }
}


////////////////////////////////////////////////////////////////////////////
// thread length

retval = uc1605("Select thread length", ip2, thread_length_flag, 2, length);

if ((retval==1)||(retval==2)) // back or cancel
{
  printf("\nERROR hole type\n");
  return -1;
}

else if (retval == 3)   //ok
{
  if (length[0] == 1 && length[1] == 0)
  {
    thread.length_flag = UF_MODL_FULL_THREAD;
  }
  if (length[0] == 0 && length[1] == 1)
  {
    thread.length_flag = UF_MODL_FIXED_LENGTH;

    int length_array_dimension = 1;
    int length_int_value[] = {1};
    int length_variable_type[] = {301};
    char * thread_length_message = "Enter thread length";
    char length_menu_list[][16] = {"Lenght"};
    char length_string_value[][31] = {"1.0000"};
    double length_double_value[] = {1.0};
    bool valid_input = false;

    do
    {
      retval = uc1613(   // show dialog-box
        thread_length_message,
        length_menu_list,
        length_array_dimension,
        length_int_value,
        length_double_value,
        length_string_value,
        length_variable_type
        );

      switch (retval)
      {
        case 1: return -1;     break;  // back
        case 2: return -1;     break;  // cancel
        case 3:              // OK - no user input
        {
          thread_length_message = "You did not enter anything!!! Enter Pocket parameters!!!";
          valid_input = false;
          break;
        }
        case 4:              // OK - user input
        {
          valid_input = true;
```

```
          break;
        }
        case 8: return -2;     break;  // error, unable to bring up dialog
        default:break;
      }

    }
    while (!valid_input);

    thread.length = length_string_value[0]; // set thread length
  }
}

////////////////////////////////////////////////////////////////////////////
// thread diameters, pitch, angle
int param_array_dimension = 4;
int param_int_value[] = {1, 1, 1, 1};
int param_variable_type[] = {301, 301, 301, 301};
char * thread_parameters = "Enter thread parameters";
char param_menu_list[][16] = {"Major Diameter","Minor Diameter", "Pitch", "Angle"};
char param_string_value[][31] = {"1.0000", "1.0000", "1.0000", "1.0000"};
double param_double_value[] = {1.0, 1.0, 1.0, 1.0};

retval = uc1613(   // show dialog-box
    thread_parameters,
    param_menu_list,
    param_array_dimension,
    param_int_value,
    param_double_value,
    param_string_value,
    param_variable_type
    );

if ((retval==1)||(retval==2)) // back or cancel
{
  printf("\nERROR hole type\n");
  return -1;
}

else if (retval == 4)    //ok
{
  thread.major_dia = param_string_value[0];
  thread.minor_dia = param_string_value[1];
  thread.pitch   = param_string_value[2];
  thread.angle   = param_string_value[3];
}

////////////////////////////////////////////////////////////////////////////
// thread taper
////////////////////////////////////////////////////////////////////////////
thread.tapped_dia = NULL;

retval = uc1605("Select thread.......", ip2, thread_tapered, 2, taper);

if ((retval==1)||(retval==2)) // back or cancel
{
  printf("\nERROR hole type\n");
  return -1;
}

else if (retval == 3)    //ok
{
  if (taper[0] == 1 && taper[1] == 0)
  {
    char *stopstring;
    char *dia = (char*) malloc(7);
    double major_dia;
    double minor_dia;
    double taper = NULL;

    thread.tapered = UF_MODL_NON_TAPERED;

    major_dia = strtod( thread.major_dia, &stopstring );
    minor_dia = strtod( thread.minor_dia, &stopstring );

    taper = ((minor_dia+major_dia)/2);
    _gcvt(taper, 7, dia);
    thread.tapped_dia = dia;
  }

  if (taper[0] == 0 && taper[1] == 1)
  {
    int taper_array_dimension = 1;
```

```
    int taper_int_value[] = {1};
    int taper_variable_type[] = {301};
    char * thread_taper_message = "Enter thread....";
    char taper_menu_list[][16] = {"Tapped Diameter"};
    char taper_string_value[][31] = {"1.0000"};
    double taper_double_value[] = {1.0};

    thread.tapered = UF_MODL_TAPERED;

    retval = uc1613(   // show dialog-box
      thread_taper_message,
      taper_menu_list,
      taper_array_dimension,
      taper_int_value,
      taper_double_value,
      taper_string_value,
      taper_variable_type
      );

    thread.tapped_dia = taper_string_value[0];
  }
}

////////////////////////////////////////////////////////////////////////////////
// thread number of starts
int starts_array_dimension = 1;
int starts_int_value[] = {1};
int starts_variable_type[] = {301};
char * thread_starts = "Enter thread number of starts";
char starts_menu_list_starts[][16] = {"Number Starts"};
char starts_string_value[][31] = {"1.0000"};
char * stopstring;
double starts_double_value[] = {1.0};

////////////////////////////////////////////////////////////////////////////////
// show dialog-box
retval = uc1613(
      thread_starts,
      starts_menu_list_starts,
      starts_array_dimension,
      starts_int_value,
      starts_double_value,
      starts_string_value,
      starts_variable_type
      );

thread.num_starts = strtod( starts_string_value[0], &stopstring );// number of starts as a double

////////////////////////////////////////////////////////////////////////////////
// thread method

int method[4];
char thread_method[][38] = {"Cut", "Rolled", "Ground", "Milled"};

retval = uc1605("Select thread method", ip2, thread_method, 4, method);

if ((retval==1)||(retval==2)) // back or cancel
{
  printf("\nERROR hole type\n");
  return -1;
}

else if (retval == 3)   //ok
{
  if (method[0] == 1 && method[1] == 0 && method[2] == 0 && method[3] == 0)
  {
    thread.method = "CUT";
  }
  if (method[0] == 0 && method[1] == 1 && method[2] == 0 && method[3] == 0)
  {
    thread.method = "ROLLED";
  }
  if (method[0] == 0 && method[1] == 0 && method[2] == 1 && method[3] == 0)
  {
    thread.method = "GROUND";
  }
  if (method[0] == 0 && method[1] == 0 && method[2] == 0 && method[3] == 1)
  {
    thread.method = "MILLED";
  }
}

thread.form = NULL;
```

```
  thread.callout = NULL;

  retval = UF_MODL_create_symb_thread(&thread, &thread_id);    // create thread
  feature_id = thread_id;

  return 0;
}


// USER DIALOG TO SELECT THE PLACEMENT FACE

int F_Thread::select_cylindrical_face(char *message, tag_t &face_id)
{
  int            retval         = 0;
  int            sel_response     = 0;
  tag_t          sel_view_id;
  double           sel_cursor_position[3] = {0., 0., 0.};
  double *         sel_cursor_position_p  = sel_cursor_position;
  UF_UI_mask_t       mask;
  UF_UI_selection_options_p_t  sel_options      = new UF_UI_selection_options_t;

  sel_options->other_options        = 0;
  sel_options->reserved           = NULL;
  sel_options->num_mask_triples       = 1;
  sel_options->mask_triples         = &mask;
  sel_options->mask_triples->object_type    = UF_solid_type;
  sel_options->mask_triples->object_subtype = UF_solid_face_subtype;
  sel_options->mask_triples->solid_type   = UF_UI_SEL_FEATURE_CYLINDRICAL_FACE;
  sel_options->scope            = UF_UI_SEL_SCOPE_WORK_PART;

  retval = UF_UI_select_single(
    message,
    sel_options,
    &sel_response,
    &face_id,
    sel_cursor_position_p,
    &sel_view_id);

  delete sel_options;

  // check return values from selecting the face
  if (sel_response == 4 || sel_response == 5) // 4: selected by name, 5: selected
  {
    retval = UF_DISP_set_highlight(face_id, 0);
  }
  else
  {
    return -1; // no valid selection, exit function with error
  }

  return 0;
}


// VALIDATES THE FEATURE

bool F_Thread::validate()
{
// ADD CODE HERE !!!
return true;
}


// SETS NAMES OF THE FACES

int F_Thread::set_face_names()
{
  return 0;
}


// STORES FACE INTERACTING FEATURES IN GLOBAL ATTRIBUTE faceint_features

uf_list_p_t F_Thread::get_faceint_features()
{
  return 0;
}
```

K.2.19 F_Face.cpp

```
// F_Face

// class for material faces

#include "F_Face.h"
#include "F_TOOLS.h"


// CONSTRUCTOR

F_Face::F_Face(tag_t face_id)
{
  printf("F_Face\n");

  this->face_id = face_id;
}


// DESTRUCTOR

F_Face::~F_Face()
{
  printf("~F_Face\n");
}


// TRANSFERS AND RETURNS PARAMETERS TO CREATE THE STEP INSTANCE

CTransferObject* F_Face::get_step_param()
{
  int retval;

  CTransferObject* param;
  param = new CTransferObject();

  int num_points=18;
  int i,j;
  double point[3],uv_box[4],uv_para[2];
  double u1[3],v1[3],u2[3],v2[3],unor[3],radii[2];
  tag_t point_id;
  double min_dist,pt_on_ent1[3],pt_on_ent2[3],schrittweite;

  int numu1,numv1,ordu1,ordv1,closedstatu1,closedstatv1,periostatu1,periostatv1;
  double uperiod1,vperiod1,knotu1[150],knotv1[150],polarray1[3000];

  int type;

  int norm_dir;
  double dir[3], box[6], radius, rad_data;

  retval = UF_MODL_ask_face_data(this->face_id, &type, point, dir, box, &radius, &rad_data,
&norm_dir);

  if (type == UF_bounded_plane_subtype)
  {
    param->IntParam(face_id, "Face_Id");
    param->DoubleArray(point, "Location", 3);
    param->DoubleArray(dir, "Normal", 3);
    param->Type("FACE_SURFACE");
    param->Subtype("PLANE_SURFACE");
  }

  else if (type == UF_cylinder_subtype)
  {
    param->IntParam(face_id, "Face_Id");
    param->DoubleArray(point, "Location", 3);
    param->DoubleArray(dir, "Axis", 3);
    param->IntParam(norm_dir, "Normal");
    param->DoubleParam(radius, "Radius");
    param->Type("FACE_SURFACE");
    param->Subtype("CYL_SURFACE");

  }
  else if (type == UF_sphere_subtype)
  {
    param->IntParam(face_id, "Face_Id");
    param->DoubleArray(point, "Center ", 3);
    param->Type("FACE_SURFACE");
    param->Subtype("SPHERE_SURFACE");
  }
```

```
  else if (type == UF_cone_subtype)
  {
    param->IntParam(face_id, "Face_Id");
    param->DoubleArray(point, "Location", 3);
    param->DoubleArray(dir, "Axis", 3);
    param->DoubleParam(radius, "Major_Rad");
    param->DoubleParam(rad_data, "Minor_Rad");
    param->IntParam(norm_dir, "Normal");
    param->Type("FACE_SURFACE");
    param->Subtype("CONE_SURFACE");
  }

  else if (type == UF_fillet_surface_subtype)
  {
    param->IntParam(face_id, "Face_Id");
    param->Type("FACE_SURFACE");
    param->Subtype("FILLET_SURFACE");
  }

  else if (type == UF_surface_of_revolution_subtype)
  {
    param->IntParam(face_id, "Face_Id");
    param->Type("FACE_SURFACE");
    param->Subtype("REVOLUTION_SURFACE");
    param->DoubleArray(dir, "Axis", 3);
    param->DoubleArray(point, "Location", 3);
  }

  else if (type == UF_b_surface_subtype)
  {
    FTN(uf5404)(&face_id,uv_box);
    schrittweite = (1/((double)num_points-1));

    for(i=(num_points)/2; i<num_points; i++)
    {
      uv_para[0]=(uv_box[0] + uv_box[1]) * ( schrittweite * i);

      for(j=2; j<num_points; j++)
      {
        uv_para[1] = (uv_box[2] + uv_box[3]) * (schrittweite * j);
        UF_MODL_ask_face_props(this->face_id,uv_para,point,u1,v1,u2,v2,unor,radii);      //     input:
face_id, uv_para, output : point, Nbleitungen, Normalenvektor, und Krümmungen
        UF_CURVE_create_point(point,&point_id);
        UF_MODL_ask_minimum_dist(point_id,this-
>face_id,1,point,1,pt_on_ent2,&min_dist,pt_on_ent1,pt_on_ent2);
        if (min_dist > 0.5)     // lösche Punkt, wenn er nicht auf der face liegt
        {
          UF_OBJ_delete_object(point_id);
          UF_CURVE_create_point(pt_on_ent2,&point_id);
          point[0]=pt_on_ent2[0];
          point[1]=pt_on_ent2[1];
          point[2]=pt_on_ent2[2];
        }
        i=num_points;
        j=num_points;
      }
    }


    param->Type("FACE_SURFACE");
    param->Subtype("NURBS_SURFACE");

    param->IntParam(this->face_id, "Face_Id");

    param->DoubleArray(point, "Location", 3);

    if (this->face_id != NULL_TAG)
    {
      FTN(uf5441)(&face_id,&numu1,&numv1,&ordu1,&ordv1,
        &closedstatu1,&closedstatv1,&periostatu1,&uperiod1,
        &periostatv1,&vperiod1,knotu1,knotv1,polarray1);


      param->IntParam(this->face_id,"Surf_Id");
      param->IntParam(numu1, "U_Poles");
      param->IntParam(numv1, "V_Poles");
      param->IntParam(ordu1, "U_Degree");
      param->IntParam(ordv1, "V_Degree");
      param->IntParam(closedstatu1, "U_Closed");
      param->IntParam(closedstatv1, "V_Closed");
      param->IntParam(periostatu1, "U_Period_Status");
      param->DoubleParam(uperiod1, "U_Period");
      param->IntParam(periostatv1, "V_Period_Status");
```

```
      param->DoubleParam(vperiod1, "V_Period");

      for (i = 0; i < (numu1 + ordu1); i++)
        param->DoubleParam(knotu1[i],"U_Knots");
      for (i = 0; i < (numv1 + ordv1); i++)
        param->DoubleParam(knotv1[i],"V_Knots");
      for (i = 0; i < 4*(numu1 * numv1); i++)
        param->DoubleParam(polarray1[i],"Poles");
    }
  }

  CStepUG* EdgeRef;
  tag_t act_edge;
  uf_list_p_t  edge_list;
  int list_count;
  CPropertyList* llist;
  uf_loop_p_t loop_list;

  retval = UF_MODL_ask_face_loops(this->face_id, &loop_list);

  while (loop_list)
  {
      CTransferObject* loop_gf = new CTransferObject();


      edge_list = loop_list->edge_list;

      UF_MODL_ask_list_count(edge_list,&list_count);

      for ( i=0; i<list_count;i++)
      {
        UF_MODL_ask_list_item(edge_list,i,&act_edge);

        EdgeRef = this->read_curve(act_edge, "Edge");

        loop_gf->Reference(EdgeRef, "Edge");
      }

      llist = loop_gf->Properties();
      llist->Name("loops");
      param->DoubleField("LOOPS", llist);

    loop_list = loop_list->next;
  }

  return param;
}


// CREATES CURVE IN STEP

CStepUG* F_Face::read_curve(tag_t edge_id, char* name)
{
  CStepUG* result;

//  if curve already exists
//  {
//    result = existing_curve
//  }
//  else
  {

    UF_CURVE_spline_s data_structure;
    UF_CURVE_line_t line_coords;
    UF_CURVE_arc_t arc_coords;
    UF_CURVE_struct_p_t  curve_struct;


    double matrix_values[9];

    double point1[3], point2[3];
    int vertex_count;

    int return_status=0,type=0,subtype=0,list_count=0,one=1,two=2,count=0;
    int error_flag=0;
    tag_t act_curve=NULL_TAG;

    //  UF_CURVE_spline_t data_structure[15];
    double tolerance=0.02;
    char error[132];

    double direction[3], unit_vec[3];
    int curve_type;
```

```
    double* curve_data;
    CStepUG* PointRef1, *PointRef2;
    double magnitude;
    tag_t search_edge = NULL_TAG;

    CTransferObject* tr_obj = new CTransferObject();


    UF_MODL_create_curve_from_edge(edge_id,&act_curve);
    UF_OBJ_set_blank_status(edge_id,UF_OBJ_BLANKED);


    return_status = UF_CURVE_ask_curve_struct(act_curve,&curve_struct);
    return_status = UF_CURVE_ask_curve_struct_data(curve_struct,
                   &curve_type,
                   &curve_data);
    tr_obj->Type("EDGE_CURVE");
    tr_obj->IntParam(edge_id, "Edge_Id");

    return_status = UF_MODL_ask_edge_verts(edge_id, point1, point2, &vertex_count);

    tr_obj->IntParam(vertex_count, "Vertex_count");

    if (vertex_count > 0)
    {
      PointRef1 = this->read_point(point1, "StartPoint");
      tr_obj->Reference(PointRef1, "StartPoint");

      PointRef2 = this->read_point(point2, "EndPoint");
      tr_obj->Reference(PointRef2, "EndPoint");

    }

    if (curve_type == UF_spline_type)
    {
      return_status=UF_CURVE_ask_spline_data(act_curve,&data_structure);

      if(return_status)
      {
        UF_get_fail_message(return_status,error);
        printf(" %s\n",error);
      }
      else
      {
        tr_obj->IntParam(act_curve, "Curve_Id");
        tr_obj->IntParam(data_structure.num_poles,"Num_Poles");
        tr_obj->IntParam(data_structure.order,"Degree");
        tr_obj->IntParam(data_structure.is_rational,"Rational");
        tr_obj->DoubleArray(data_structure.knots,                                    "Knots",
data_structure.num_poles+data_structure.order);

        count=0;

        for (int j=0; j<data_structure.num_poles ; j++)
        {
          tr_obj->DoubleParam(data_structure.poles[j][0], "Poles");
          tr_obj->DoubleParam(data_structure.poles[j][1], "Poles");
          tr_obj->DoubleParam(data_structure.poles[j][2], "Poles");
          tr_obj->DoubleParam(data_structure.poles[j][3], "Poles");

        }
        tr_obj->DoubleParam(data_structure.start_param,"Start_Param");
        tr_obj->DoubleParam(data_structure.end_param,"End_Param");
        tr_obj->Subtype("NURBS_CURVE");
      }
    }
    else if (curve_type == UF_line_type)
    {
      return_status = UF_CURVE_ask_line_data(act_curve, &line_coords);

      if(return_status)
      {
        UF_get_fail_message(return_status,error);
        printf(" %s\n",error);
      }
      else
      {
        for (int i = 0; i< 3;i++)
          direction[i] = line_coords.end_point[i]-line_coords.start_point[i];
        tolerance = 0.0001;
        return_status = UF_VEC3_unitize(direction, tolerance, &magnitude, unit_vec);
        if (return_status) // normalization was not possible
          tr_obj->DoubleArray(direction, "Direction",3);
```

```
        else
          tr_obj->DoubleArray(unit_vec, "Direction",3);
        tr_obj->IntParam(act_curve, "Curve_Id");
        tr_obj->Subtype("LINE_CURVE");

      }
    }
    else if (curve_type == UF_circle_type)
    {
      return_status = UF_CURVE_ask_arc_data(act_curve, &arc_coords);

      if(return_status)
      {
        UF_get_fail_message(return_status,error);
        printf(" %s\n",error);
      }
      else
      {

        return_status = UF_CSYS_ask_matrix_values(arc_coords.matrix_tag, matrix_values);
        if(return_status)
        {
          UF_get_fail_message(return_status,error);
          printf(" %s\n",error);
        }

        tr_obj->DoubleArray(matrix_values, "Matrix",9);
        tr_obj->DoubleParam(arc_coords.start_angle, "StartAngle");
        tr_obj->DoubleParam(arc_coords.end_angle, "EndAngle");
        tr_obj->DoubleArray(arc_coords.arc_center, "Center",3);
        tr_obj->DoubleParam(arc_coords.radius, "Radius");
        tr_obj->IntParam(act_curve, "Curve_Id");
        tr_obj->Subtype("ARC_CURVE");
      }
    }

    result = IMCreateFeature(tr_obj);

  }

  return result;
}


// CREATES POINT IN STEP

CStepUG* F_Face::read_point(double point_coords[3], char* name)
{
  CStepUG* result;

//  if point already exists
//  {
//    result = existing_point
//  }
//  else
  {
    tag_t point = NULL_TAG, search_point = NULL_TAG;
    int response;

    CTransferObject* tr_obj = new CTransferObject();
    tr_obj->Type("VERTEX_POINT");
    tr_obj->Subtype(name);
    response = UF_CURVE_create_point(point_coords, &point);
    tr_obj->IntParam(point, "Point_Id");
    response = UF_OBJ_set_blank_status(point,UF_OBJ_BLANKED);
    tr_obj->DoubleArray(point_coords, name, 3);
    result = IMCreateFeature(tr_obj);
  }

  return result;
}


// ADDS USER SPECIFIED DOUBLE ATTRIBUTE TO FACE

F_Face::add_ug_attr_double(char* name, double value)
{
  UF_ATTR_value_t uf_value;

  uf_value.type = UF_ATTR_real;
  uf_value.value.real = value;

  UF_ATTR_assign(this->face_id, name, uf_value);
```

```
}


// READS USER SPECIFIED DOUBLE ATTRIBUTE FROM FACE

double F_Face::read_ug_attr_double(char* name)
{
  double result;

  UF_ATTR_value_t uf_value;
  int retval;

  retval = UF_ATTR_read_value(this->face_id, name, UF_ATTR_real, &uf_value);

  if (uf_value.type != 0)
  {
    result = uf_value.value.real;
  }
  else
  {
    result = 0.0;
  }

  return result;
}


// ADDS USER SPECIFIED INTEGER ATTRIBUTE TO FACE

F_Face::add_ug_attr_int(char* name, int value)
{
  UF_ATTR_value_t uf_value;

  uf_value.type = UF_ATTR_integer;
  uf_value.value.integer = value;

  UF_ATTR_assign(this->face_id, name, uf_value);
}


// READS USER SPECIFIED ATTRIBUTE FROM FACE

int F_Face::read_ug_attr_int(char* name)
{
  int result;

  UF_ATTR_value_t uf_value;
  int retval;

  retval = UF_ATTR_read_value(this->face_id, name, UF_ATTR_integer, &uf_value);

  if (uf_value.type != 0)
  {
    result = uf_value.value.integer;
  }
  else
  {
    result = 0;
  }

  return result;
}
```

K.2.20 F_Tools.cpp

```
// F_Tools

// some global functions for general tasks

#include "F_TOOLS.h"


// CREATES AND RETURNS INSTANCE OF FESTEVAL FEATURE CLASS (TYPE KNOWN)

F_Feature* F_TOOLS_get_feature_object(F_Type type)
{
  F_Feature* result = NULL;

  switch(type)
  {
  case F_UNKNOWN :
```

```
    result = NULL;
    break;

  case F_INSTANCE :

    result = NULL;
    break;

  case F_CHAMFER :

    result = new F_Chamfer();
    break;

  case F_PLANAR_FACE :

    result = new F_Planar_Face();
    break;

  case F_POCKET :

    result = new F_Pocket();
    break;

  case F_HOLE_SIMPLE :

    result = new F_Hole_Simple();
    break;

  case F_HOLE_CBORE :

    result = new F_Hole_CBore();
    break;

  case F_HOLE_CSUNK :

    result = new F_Hole_CSunk();
    break;

  case F_SLOT :

    result = new F_Slot();
    break;

  case F_THREAD :

    result = new F_Thread();
    break;

// ADD CASES FOR NEW FEATURE TYPES HERE

  }

  return result;
}


// CREATES AND RETURNS INSTANCE OF FESTEVAL FEATURE CLASS (UG FEATURE KNOWN)

F_Feature* F_TOOLS_get_feature_object(tag_t feature_id)
{
  F_Feature* result = NULL;

  F_Type type;
  type = F_TOOLS_get_feature_type(feature_id);

  switch(type)
  {
  case F_UNKNOWN :

    result = NULL;
    break;

  case F_INSTANCE :            // IF INSTANCE USE MASTER (RECURSION !)

    tag_t master_feature;
    UF_MODL_ask_master(feature_id, &master_feature);
    result = F_TOOLS_get_feature_object(master_feature);

    break;

  case F_CHAMFER :

    result = new F_Chamfer(feature_id);
```

```
    break;

  case F_PLANAR_FACE :

    result = new F_Planar_Face(feature_id);
    break;

  case F_POCKET :

    result = new F_Pocket(feature_id);
    break;

  case F_HOLE_SIMPLE :

    result = new F_Hole_Simple(feature_id);
    break;

  case F_HOLE_CBORE :

    result = new F_Hole_CBore(feature_id);
    break;

  case F_HOLE_CSUNK :

    result = new F_Hole_CSunk(feature_id);
    break;

  case F_SLOT :

    result = new F_Slot(feature_id);
    break;

  case F_THREAD :

    result = new F_Thread(feature_id);
    break;

// ADD CASES FOR NEW FEATURE TYPES HERE

  }

  return result;
}


// CREATES AND RETURNS INSTANCE OF FESTEVAL FEATURE CLASS (STEP PARAMS KNOWN)

F_Feature* F_TOOLS_get_feature_object(CTransferObject* param)
{
  F_Feature* result = NULL;

  F_Type type;
  type = F_TOOLS_get_feature_type(param->Type());

  switch(type)
  {
  case F_UNKNOWN :

    result = NULL;
    break;

  case F_INSTANCE :

    result = NULL;
    break;

  case F_CHAMFER :

    result = new F_Chamfer(param);
    break;

  case F_PLANAR_FACE :

    result = new F_Planar_Face(param);
    break;

  case F_POCKET :

    result = new F_Pocket(param);
    break;

  case F_HOLE_SIMPLE :
```

```
    result = new F_Hole_Simple(param);
    break;

  case F_HOLE_CBORE :

    result = new F_Hole_CBore(param);
    break;

  case F_HOLE_CSUNK :

    result = new F_Hole_CSunk(param);
    break;

  case F_SLOT :

    result = new F_Slot(param);
    break;

  case F_THREAD :

    result = new F_Thread(param);
    break;

// ADD CASES FOR NEW FEATURE TYPES HERE

  }

  return result;
}


// RETURNS TYPE OF FEATURE (UG FEATURE KNOWN)

F_Type F_TOOLS_get_feature_type(tag_t feature_id)
{
  F_Type result = F_UNKNOWN;
  char* type;

  UF_MODL_ask_feat_type(feature_id, &type);

  result = F_TOOLS_get_feature_type(type);

  return result;
}


// RETURNS TYPE OF FEATURE (UG TYPE KNOWN)

F_Type F_TOOLS_get_feature_type(char* type)
{
  F_Type result;

  if (strcmp(type, "INSTANCE") == 0)
  {
    result = F_INSTANCE;
  }
  else if (strcmp(type, "CHAMFER") == 0)
  {
    result = F_CHAMFER;
  }
  else if (strcmp(type, "SWP104") == 0)
  {
    result = F_PLANAR_FACE;
  }
  else if (strcmp(type, "RECT_POCKET") == 0)
  {
    result = F_POCKET;
  }
  else if (strcmp(type, "SIMPLE HOLE") == 0)
  {
    result = F_HOLE_SIMPLE;
  }
  else if (strcmp(type, "CBORE_HOLE") == 0)
  {
    result = F_HOLE_CBORE;
  }
  else if (strcmp(type, "CSUNK_HOLE") == 0)
  {
    result = F_HOLE_CSUNK;
  }
  else if (strcmp(type, "RECT_SLOT") == 0)
  {
    result = F_SLOT;
```

```
  }
  else if (strcmp(type, "SYMBOLIC_THREAD") == 0)
  {
    result = F_THREAD;
  }

// ADD CASES FOR NEW FEATURE TYPES HERE

  else
  {
    result = F_UNKNOWN;
  }

  return result;
}


// CONVERTS STRING-VALUE TO DOUBLE-VALUE

double F_TOOLS_str2dbl(const char* string)
{
  char *result_string;
  double result;

  if (string != NULL)
  {
    result_string = strstr(string, "=");
    if (result_string != NULL)
    {
      result_string = _strinc(result_string);
      result = atof(result_string);
    }
    else
    {
      result = 0.0;
    }
  }
  else
  {
    result = 0.0;
  }

  return result;
}


// GET AND RETURN ALL FEATURES (RECURSIVE FUNCTION)

uf_list_p_t F_TOOLS_get_feature_list(tag_t base)
{

// GET CHILDREN OF BASE_FEATURE

  uf_list_p_t feature_list;

  int num_parents;
  tag_t* parent_array;
  int num_children;
  tag_t* children_array;

  UF_MODL_ask_feat_relatives(  base,
               &num_parents,
               &parent_array,
               &num_children,
               &children_array);

// TRANSFER ARRAY TO LIST

  int f_nr;

  UF_MODL_create_list(&feature_list);

  for (f_nr = 0; f_nr < num_children; f_nr++)
  {
    tag_t feature_id;
    feature_id = children_array[f_nr];

    UF_MODL_put_list_item(feature_list, feature_id);

// GET CHILDREN AND ADD TO LIST

    uf_list_p_t children_list;
    int children_count;
```

```
    children_list = F_TOOLS_get_feature_list(feature_id);    // RECUSRION !!!!
    UF_MODL_ask_list_count(children_list, &children_count);

    int c_nr;

    for (c_nr = 0; c_nr < children_count; c_nr++)
    {
      tag_t children_id;
      UF_MODL_ask_list_item(children_list, c_nr, &children_id);

      UF_MODL_put_list_item(feature_list, children_id);
    }
  }

// ORDER LIST

  uf_list_p_t ordered_feature_list;
  UF_MODL_create_list(&ordered_feature_list);

  ordered_feature_list = F_TOOLS_order_uf_list(feature_list);

  return ordered_feature_list;
}


// ORDERS A LIST WITH UG FEATURES (TAG_T)

uf_list_p_t F_TOOLS_order_uf_list(uf_list_p_t feature_list)
{
  uf_list_p_t ordered_feature_list;
  UF_MODL_create_list(&ordered_feature_list);

  int count;
  UF_MODL_ask_list_count(feature_list, &count);

 // GET LOWEST ENTRY

  if (count > 0)
  {
    tag_t lowest;

    UF_MODL_ask_list_item(feature_list, 0, &lowest);

    for (int f_nr = 0; f_nr < count; f_nr++)
    {
      tag_t actual;
      UF_MODL_ask_list_item(feature_list, f_nr, &actual);

      if (F_TOOLS_get_time_stamp(actual) < F_TOOLS_get_time_stamp(lowest))
      {
        lowest = actual;
      }
    }

 // PUT LOWEST IN ORDERED LIST

    UF_MODL_put_list_item(ordered_feature_list, lowest);

 // DELETE LOWEST ENTRY FROM ORIGINAL LIST

    UF_MODL_delete_list_item(&feature_list, lowest);

 // ORDER REST OF ORIGINAL LIST

    uf_list_p_t ordered_rest_list;

    ordered_rest_list = F_TOOLS_order_uf_list(feature_list);

 // ATTACH ORDERED REST LIST TO ORDERED LIST

    UF_MODL_ask_list_count(ordered_rest_list, &count);

    for (f_nr = 0; f_nr < count; f_nr++)
    {
      tag_t actual;
      UF_MODL_ask_list_item(ordered_rest_list, f_nr, &actual);

      UF_MODL_put_list_item(ordered_feature_list, actual);
    }
  }

  return ordered_feature_list;
```

```
}


// RETURNS THE TIME STAMP OF A FEATURE (ORDER OF CREATION)

int F_TOOLS_get_time_stamp(tag_t feature_id)
{
  char* name;
  char* timestring;
  char* compstr = "()";
  int timestamp;

  UF_MODL_ask_feat_name(feature_id, &name); // GET FEATURE NAME

  int pos = 0;              // FIND OPEN BRACKET
  while (name[pos] != compstr[0])
  {
    pos = pos + 1;
  }
  pos = pos + 1;

  timestring = &name[pos];

  pos = 0;                // FIND CLOSED BRACKET
  while (timestring[pos] != compstr[1])
  {
    pos = pos + 1;
  }
  timestring[pos] = compstr[2];

  timestamp = atoi(timestring);       // CONVERT TO INT

  return timestamp;
}


// CREATES LABELED DIALOG WITH THE OPTIONS YES AND NO

bool F_TOOLS_yes_no_dialog(char* message)
{
  bool result;
  int retval;

  char label[1][38] = {""};

  for (unsigned int pos=0; pos <= strlen(message); pos++)
  {
    label[0][pos] = message [pos];
  }

  retval = uc1603( "Please select OK or Cancel.", 1,
             label,
             1);

  if (retval == 5)
  {
    result = true;
  }
  else
  {
    result = false;
  }

  return result;

}
```

K.3 UG menu file

K.3.1 Festeval_UG.men

```
VERSION 120

    EDIT UG_GATEWAY_MAIN_MENUBAR

    BEFORE UG_HELP
      CASCADE_BUTTON UISTYLER_DLG_CASCADE_BTN
      LABEL FESTEVAL APPLICATIONS
    END_OF_BEFORE

    MENU UISTYLER_DLG_CASCADE_BTN
      BUTTON FESTEVAL_BTN
      LABEL Festeval Environment
      ACTIONS FESTEVAL_UG.dlg

      BUTTON STEP_BTN
      LABEL Step Processor
      ACTIONS STEP_UG.dlg
    END_OF_MENU
```