Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Research Report
RR-91-20

FEAT-REP:

Representing Features in CAD/CAM

Christoph Klauck,
Ansgar Bernardi,
Ralf Legleitner

June 1991

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Postfach 20 80
D-6750 Kaiserslautern, FRG
Tel.: (+49 631) 205-3211/13
Fax: (+49 631) 205-3210

Stuhlsatzenhausweg 3
D-6600 Saarbrücken 11, FRG
Tel.: (+49 681) 302-5252
Fax: (+49 681) 302-5341

Deutsches Forschungszentrum für Künstliche Intelligenz

The German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) with sites in Kaiserslautern und Saarbrücken is a non-profit organization which was founded in 1988 by the shareholder companies ADV/Orga, AEG, IBM, Insiders, Fraunhofer Gesellschaft, GMD, Krupp-Atlas, Mannesmann-Kienzle, Nixdorf, Philips and Siemens. Research projects conducted at the DFKI are funded by the German Ministry for Research and Technology, by the shareholder companies, or by other industrial contracts.

The DFKI conducts application-oriented basic research in the field of artificial intelligence and other related subfields of computer science. The overall goal is to construct *systems with technical knowledge and common sense* which - by using AI methods - implement a problem solution for a selected application area. Currently, there are the following research areas at the DFKI:

❑ Intelligent Engineering Systems
❑ Intelligent User Interfaces
❑ Intelligent Communication Networks
❑ Intelligent Cooperative Systems.

The DFKI strives at making its research results available to the scientific community. There exist many contacts to domestic and foreign research institutions, both in academy and industry. The DFKI hosts technology transfer workshops for shareholders and other interested groups in order to inform about the current state of research.

From its beginning, the DFKI has provided an attractive working environment for AI researchers from Germany and from all over the world. The goal is to have a staff of about 100 researchers at the end of the building-up phase.

Prof. Dr. Gerhard Barth
Director

FEAT-REP
Representing Features in CAD/CAM

Christoph Klauck, Ansgar Bernardi, Ralf Legleitner

DFKI-RR-91-20

A short version of this paper will be published in the Proceedings of the 4th International Symposium on Artificial Intelligence: Applications in Informatics

FEAT-REP: Representing Features in CAD/CAM

Dipl.-Inform. Christoph **Klauck**,
Dipl.-Ing. Ralf **Legleitner**
Dipl.-Inform. Ansgar **Bernardi**,

ARC-TEC Project
German Research Center for Artificial Intelligence
DFKI GmbH, Postfach 2080, D-6750 Kaiserslautern, Germany
Telefon: +49631/205-3477, 205-4068
Fax: +496ni-kl.de
bernardi@dfki.uni-kl.de
legleit@dfki.uni-kl.de

1. Abstract

When CAD/CAM experts view a workpiece, they perceive it in terms of their own expertise. These terms, called *features*, which are build upon a *syntax* (geometry) and a *semantics* (e.g. skeletal plans in manufacturing or functional relations in design), provide an abstraction mechanism to facilitate the creation, manufacturing and analysis of workpieces. Our goal is to enable experts to represent their own *feature-language* via a *feature-grammar* in the computer to build *feature-based* systems e.g. CAPP systems. The application of formal language terminology to the feature definitions facilitates the use of well-known formal language methods like parsing in conjunction with our flexible knowledge representation formalism FEAT-REP.

Keywords: feature, feature recognition, feature-language, feature-grammar, Attributed Node-Label-Controlled Graph Grammars

2. Table of Contents

3. Introduction

An important step towards truely Computer Integrated Manufacturing (CIM) is the Computer Aided Process Planning (CAPP). A CAPP system shall use the information provided by CAD (Computer Aided Design) to generate the process plan for the manufacturing of the workpiece in question by means of CAM (Computer Aided Manufacturing).

The solid modellers currently used in CAD describe a workpiece only in terms of lower-level entities like faces, edges, vertices (topology), surfaces, lines and points (geometry), or volumetric primitives like cylinders or cones (cf. [8]). While these lower-level entities represent the complete quantitative information about a

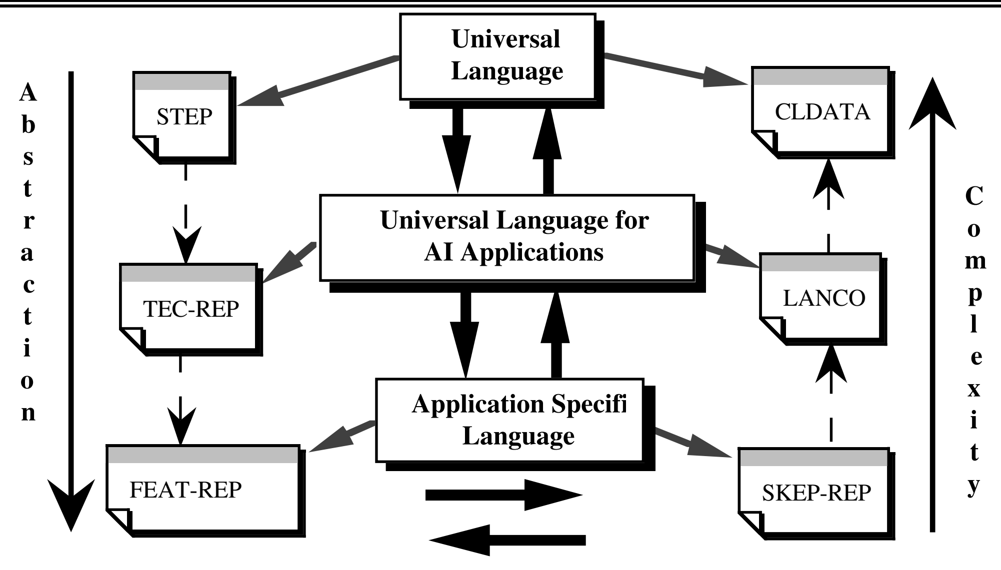


fig. 1 The basic method of PIM

workpiece, efficient planning strategies rely on higher-level (qualitative) information supporting abstract reasoning to accomplish their goals. In our approach these higher-level entities are the so-called *features* which must be extracted from the data of the CAD models [20, 15]. In the discussion about the role of solid modelling as the interface between design and manufacturing by us or e.g. Mike J. Pratt in [40] these higher-level informations build the bridge between the workpiece created by the designer and the process plan. Employing features, an experts knowledge in this domain can be suitable formalized and used in planning systems (cf. [11]).

The proposed system PIM (Planning In Manufacturing) in [11] recognizes features in a given representation of a workpiece, finds skeletal plans associated to these features, and refines these plans to the CLDATA code (Cutter Location DATA) necessary for manufacturing. This sequence of abstractions/refinements is illustrated in figure 1 and follows the expertise model of human experts (cf. [47]). To bridge the gap between the geometric description e.g. represented in STEP (STandard for the Exchange of Product model data) and the manufacturing instructions e.g. represented in CLDATA code, the sequence of representations on different abstraction levels reduces this problem (and the complexity of the problem) to the problem of finding an associated *skeletal plan* to a given workpiece described in terms of features. So representing and recognizing features is a necessary step to bridge the gap between CAD and CAM. It is important to note that in general different domains like design, turning or milling leads to different features and that a standardization of all features is just unreasonable.

In this paper we show that it is possible to describe features by means of formal languages via attributed node-label-contolled graph grammars. The area of formal languages is a well established field of research and provides a powerful set of methods like parsing and knowledge about problems, their complexity and how they could be solved efficiently. The use of formal languages for feature descriptions facilitates the application of these results to the area of feature recognition and CAPP.

4. What are Features ?

In the current literature there is no consensus on a precise definition of the term *feature*. Most researchers working in this area agree that a feature is an abstraction of lower-level design and manufacturing information which depends on the context of the machine shop [20]. Features that are required for design may differ considerably from those required for manufacturing or assembly, even though they maybe based on the same lower-level entities. Cunsulting several experts of manufacturing and design showed that these differences are reasonable.

In the first section of this chapter a short review of relevant work in literature will be done. In the second section the term feature will be defined by us under consideration of the definitions in the first section.

4.1. Review of Relevant Work

John R. Dixon and John J. Cunningham have defined a feature as "*any geometric form or entity that is used in reasoning in one or more design or manufacturing activities (i.e. fit, function, manufacturability evaluation, analysis interfacing, tool and die design, inspectability, serviceability, etc.).*" [18]. Features there originate (in bottom-up fashion) in the reasoning process used in various design and manufacturing activities. If a geometric form is used for reasoning, then that form is a feature which needs to be represented separatly. The authors of the paper stated out, that different manufacturing processes will require different features for their various activities, that is, every process-activity combination has its own set of features. In general these features have attributes according to their type. In order to compile the features needed, the authors have examined several activities in connection with several manufacturing processes. The heuristics for each process-activity pair generate a corresponding set of features. To illustrate, two examples out of their paper will be presented in which the derivation of certain features is shown from a bit of heuristic manufacturing knowledge. Within the body of knowledge governing the manufacture of aluminum extrusions, the authors find the heuristics:

a) *Long thin walls should have ribs.*

From this heuristic two features are identified: *walls* and *ribs*. The qualifier for the feature *wall* is its length-to-thickness ratio. The feature *rib* has no qualifier in this heuristic.

b) *The apex of [triangular] slots should be well rounded and the rounding radius should be at least twice the stock thickness.*

Stock thickness refers to the nominal wall thickness of the sheet, so the *wall* is the feature, and thickness is an attribute. *Slot* is a feature (in this case it is specifically a triangular slot) and the apex radius is its attribute, which is qualified relative to the stock thickness.

So as conclusion features are higher order abstract geometric forms or entities that are used in reasoning about the topology and geometry of designed artifacts during various design and manufacturing activities and the features and their qualitative and quantitative qualifiers originate in the heuristics that surround these activities.

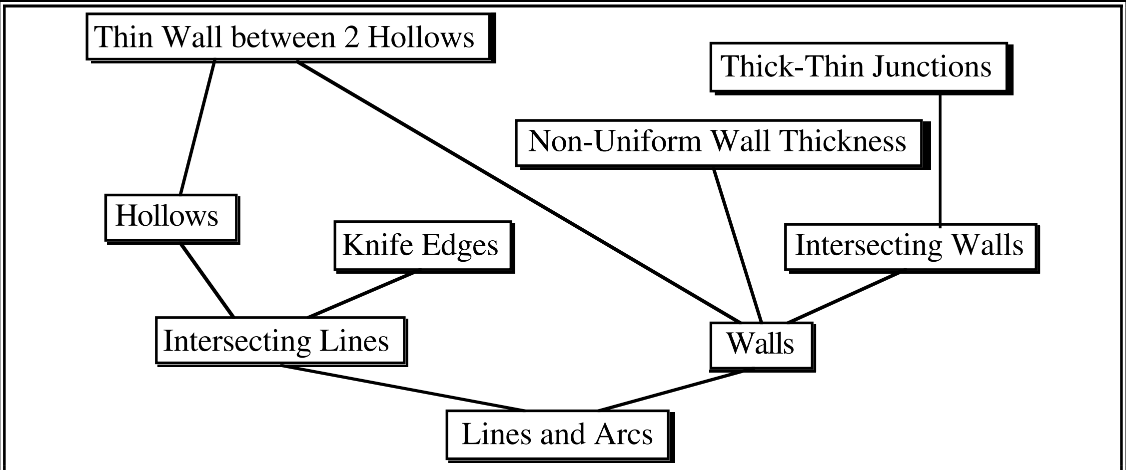


fig. ?2 A sample feature hierarchy

Another definition was given by David C. Gossard and J. K. Hirschtick: "*A feature of a geometric model in a given context is a descriptor of that model whose presence and/or size is relevant within the given context.*" [27]. The authors subdivide features into lower-level features and high-level features. The high-level features are defined in terms of intermediate features of lower-level. These intermediate features are defined in terms of other intermediate features lower than themselves, and so on.

Geometric entities, such as points, lines, arcs, splines, surfaces, and primitive solids be considered low-level features, the lowest-level, or ground state, features for a geometric feature extraction problem. An example of a hierarchy of features is shown in figure ?2.

In [27] they stated out, that the high-level features which are to be recognized depends on the function, or context, of each particular feature extraction system: No universal set of features exists which will provide a satisfactory description of a part for all applications. As conclusion features are defined in a hierarchy of features where the lowest-level of this hierarchy is build via geometric entities, the so-called lower-level features.

Tien-Chien Chang has defined a feature in his book [15] as "*a subset of geometry on an engineering part which has a special design or manufacturing characteristic.*". A feature has its specific geometry and must be associated with some feature attributes. The attributes can be dimensions, dimensional tolerance, manufacturing notes, etc. Depending on the application, different information maybe included. In any case, a feature is a geometrically independent entity. It contains some meanings useful to the application it is designed for. Based on the geometry, Tien-Chien Chang classify features into the following:

- *Face feature* – features defined by two or three dimensional faces. (e.g. gear, fillet, hexagon)
- *Volumetric features* – features defined by three dimensional, enclosed volumes. (e.g. hole, boss, simple slot, T slot, V slot, pocket, groove, cutout)

Based on the applications:

- *Design features* – features meaningful to design. (e.g. hole, chamfer, groove, countersink, screw thread)
- *Manufacturing features* – features meaningful to manufacturing. (e.g. hole, groove, hole tip, fillet, chamfer, countersink)

He stated out, that the term feature does not have a definition which is agreed upon by everyone because it is definitely application specific. So design features and manufacturing features do overlap; many features are identical and some are different, and some use the same name but carry different meaning.

A similar (informal) definition was given by J. J. Shah and M. T. Rogers ([42]); they define a feature as "*recurring patterns of information related to a part's description.*" and distinguishe features via there type in:

- *Form features*. These are groups of geometric entities that define attributes of a part´s nominal size and shape. Here also they distinguish between *primary features* and *subfeatures*; primary features can be thought of as part´s major shape, while subfeatures are alterations made to the major shapes.
- *Precision features*. These are acceptable deviations from the nominal geometry. Included in this set are dimensional tolerances and surface finish.
- *Material features*. These specify material types, grades, properties, heat treatment, surface treatments, etc.

```
Feature_lD          21210
Feature _ Name      con i cal _ hole _ r
Feature_Type        sub
Compatible_Parnt    cylinder
Compatible_Sub      groove_1
User_Def_Param      entity_num    diameter    depth
                      angle          e          theta
Expressions         # (abcos*)
                      # ( absin * )
                       # (a2/)
                       # (ab2/tan2*/)
                       # (ab2/tan2*/c + )
                       # (ab – 2/)
Derived _Param
FPV_Subtree         CYL(a1, a2, a3, f1, f2, 0, 0, 0, 0)
                          if (a1 = 1) (0, 0, 0, 0, 0, 0)
                          if (a1 = 2) (0, 0, f3, 180, 0, 0)[U]
                          CON(a1, f4, 0, f5, f2, a3, 0, 0, 0)
                          if (a1 = 1) (0, 0, 0, 0, 0, 0)
                          if (a1 = 2) (0, 0, f3, 180, 0, 0)(—)
FPV_parameters      # (e1 ((s1, a5), (s1, a6)))
                      # (e2 ((s1, a5), (s1, a6)))
                      # (p1, a2)
                      # (e3 (s1, a2))
                      # (e4 ((s1, a2), (s1, a6)))
Blank
Blank
Inherit_ Rules
Cognition_Rules     # ((p1, a2) > e5 ((s1, a2), (s1, a4), (s1, a3))
                       # ((s1, a2) < (p1, a1))
                       # ((s1, a5) < e6 ((p1, a1), (s1, a2)))
                       # ((s1, a6) < 180)
                       # ((s1, a6) > 0)
Interpret_Rules
```

fig. ?3 Feature property list of the feature conical hole

An object-oriented programming approach to representing feature descriptions leads in [42] to property lists stored in a database. Addition of a new generic feature means adding a new feature property list to the database: no code alterations are needed. These lists are organized in a frame system where property inheritance is possible between related frames. Figure ?3 is a property list for a conical hole with the property values specified. These properties provide the generic feature definitions, including means of identification, parameter definitions, the inheritance of properties and parameters, and constraints on how a feature maybe used.

Other similar definitions of features can be found in [36, 22, 14, 19, 28] and [34].

The common ground of these feature definitions is first, that they are allways based on the geometry of workpieces. Second the features get their effectiveness out of the informations associated with them. Information-less form features are sometimes defined but they become only important when informations are associated with them. Third the definitions of features and their associated informations are dependend of their application. Finally a universal set of feature definitions is not reasonable.

The differences of these feature definitions are the attributes which the features may have and the hierarchies where they are embedded. Some definitions have no attributes and some have no hierarchies. Also the classifications of the features are different. Finally their origin and the listet definitions of the features are also different in the listet literature.

4.2. Feature Definition

In our paper the term *feature* is defined as a description element based on geometrical and technological data of a product which an expert in a domain associates with certain informations. They are firstly distinguished by their kind as

- functional features, e.g. *seat of the rolling bearing* or *O-ring groove*,
- qualitative features, e.g. *bars* or *solid workpiece*,
- geometrical (form-) features, e.g. *shoulder*, *groove* or *drilled hole*,
- atomic features, e.g. *toroidal shell*, *ring*, *shape tolerance* or *surface finish*.

and they are secondly distinguished by their application as

- design features, e.g. *crank* or *coupler*,
- manufacturing features:
  - turning features, e.g. *shoulder* or *neck*,
  - milling features, e.g. *step* or *pocket*,
  - drilling features e.g. *stepped-hole* or *lowering*,
  - ...
- ...

Our definition follows the one of Tien-Chien Chang [14] and is distinguished by the emphasis on an expert in a domain. In particular every feature will be defined by a respective expert because his area, like machines, tools or the characteristics of them, and his ideas, creativity and experience, like special tricks, are included in this definition. In this sense the features can been seen as a *language* of an expert in a domain. It is important to note that this language represents the *know-how* of the expert respectively the machine shop and that this language is an individual ("expert in a domain" dependent) one. It is also important to note that such a language has a syntax and a semantics. What we interpret as syntax and semantics of these *feature-languages* will be explained in the next section. So it is incumbent upon the XPS-shells or -tools only to define a representation language for features respectively the feature-language and not the features itself; they must be defined individually for every XPS in its individual area.

In comparison with the feature definitions in the previous section the differences to our feature definition are the calssification of the features which results out of the distinction in syntax and semantics of features, and the origin of the features, in our case always experts. The common ground of our definition and the definition listet in the previous section is first that geometry serves as a basis for the feature definitions and second that the features get their effectiveness out of the informations associated with them. Finally the definitions of features (and their associated informations) are depentend of their application and in our case more restricted to the dependency of an expert.

5. Syntax and Semantics of Feature-Languages

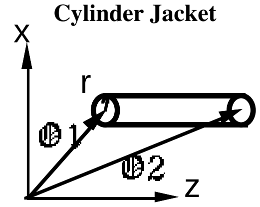
In the previous chapter we defined the term feature. There also is mentioned briefly an analogue between the feature descriptions and (formal) languages which results in the term *feature-language*. In this chapter the syntax and semantics of feature-languages will be described in general. In figure 6 the conclusion of this analogue is shown. Before we will discuss the syntax it should be pointed out that the expert chooses/creates a syntax of the features which is dependent of the informations associated with the features.

5.1. The Syntax

The first important issue about the features is that the expert bases his definitions on the boundary surfaces and the technological informations of the workpiece, like tolerances or surface finish, which are assigned to one or more surfaces. Our representation formalism TEC-REP ([12]) supplies these entities which are used as atomic features. TEC-REP also supplies a topology graph to represent the neighbourhoodness of surfaces. Some examples of these description entities are presented in figure 2.

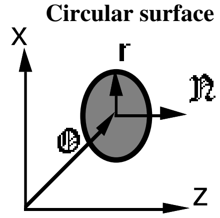
To define the geometrical features the expert uses the atomic features and the geometrical features itself. One simple example of features described by an expert is shown in figure 3 and figure 9. An example of corresponding attributed syntax rules is given in figure 4.

Cylinder Jacket



CJ(reference_point1: <vector>,
reference_point2: <vector>,
radius: <num>,
direction_of_material: [+,-])

Circular surface



C(reference_point: <vector>,
direction_vector: <vector>,
radius: <num>,
direction_of_material: [+,-])

Shape Tolerance DIN 7184

ST(surface_number: <num>,
tolerance_type: [SN, EN, RN, C, LD, PD],
tolerance_size: <num>)

Surface Finish

SF(surface_number: <num>,
surface_finish: [Rt, Rz, Rp, Ra,...],
value: <num>)

fig. 2 some entities of TEC-REP

long turning surface → cylinder jacket
flank → ring
left shoulder → flank, long turning surface
right shoulder → long turning surface, flank
groove → left shoulder, right shoulder
insertion → left shoulder, right shoulder

fig. 3 simple syntax for features

It is important to note that this kind of rules is only sufficient when features of rotational symmetric parts are described; in general graph-based rules are needed (cf. [16]) because the features will be defined in general by the topological graph of their parts. An example can be seen in figure 5.

The descriptions of functional features are based upon the descriptions of geometrical features and differ in the connection to other products. The functionality is defined via the description of the functional relation between the functional feature and one or more other products. The syntax rules of these features differ in the additional attributes which describe the functional relation and the technological restrictions. The descriptions of qualitative features are also based upon geometrical

long turning surface → cylinder jacket
lts.radius = cj.radius
long turning surface → long turning surface, insertion, long turning surface
lts0.radius = lts1.radius
lts1.radius = lts2.radius

fig. 4 attributed syntax rules

features and represent a more abstract description of a workpiece. Their syntax rules differ in the additional attributes which describe the technological and geometrical restrictions. As conclusion we can state out that the geometrical description in addition with attributes about the context, functionality and technology forms the syntax of a feature.

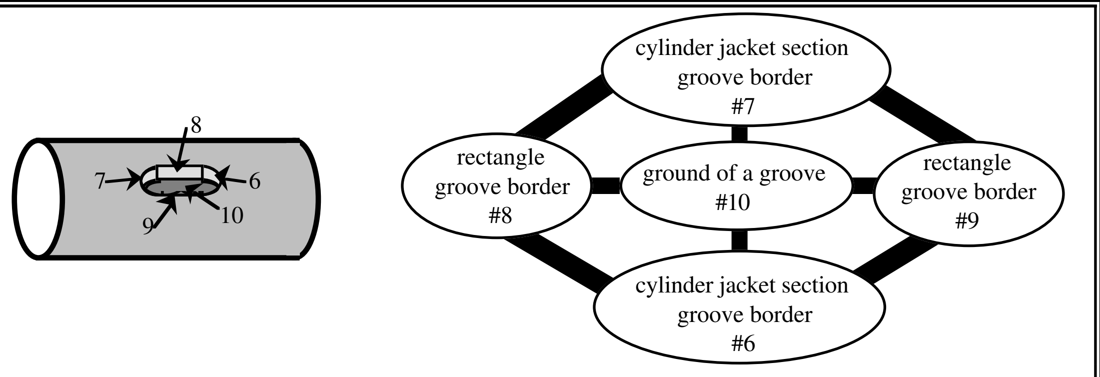


fig. 5 topology graph of a key groove

5.2. The Semantics

Now we can describe the semantics of a feature-language. The main thing of the features is, that the expert associates certain informations with the features. In this view the syntax of the features can be seen as a vessel to carry the informations, the *semantics* of the features. What kind of informations associates the expert with his features ? This depends on the domain where he works. A designer for example associates first the functionality and the costs with his features. So when he says "seat of the rolling bearing" he first describes the syntax of the feature, e.g. geometry and technology, and secondly he describes the semantics of the feature, e.g. that this part will be used as a seat. Our research concentrates on the semantics of the

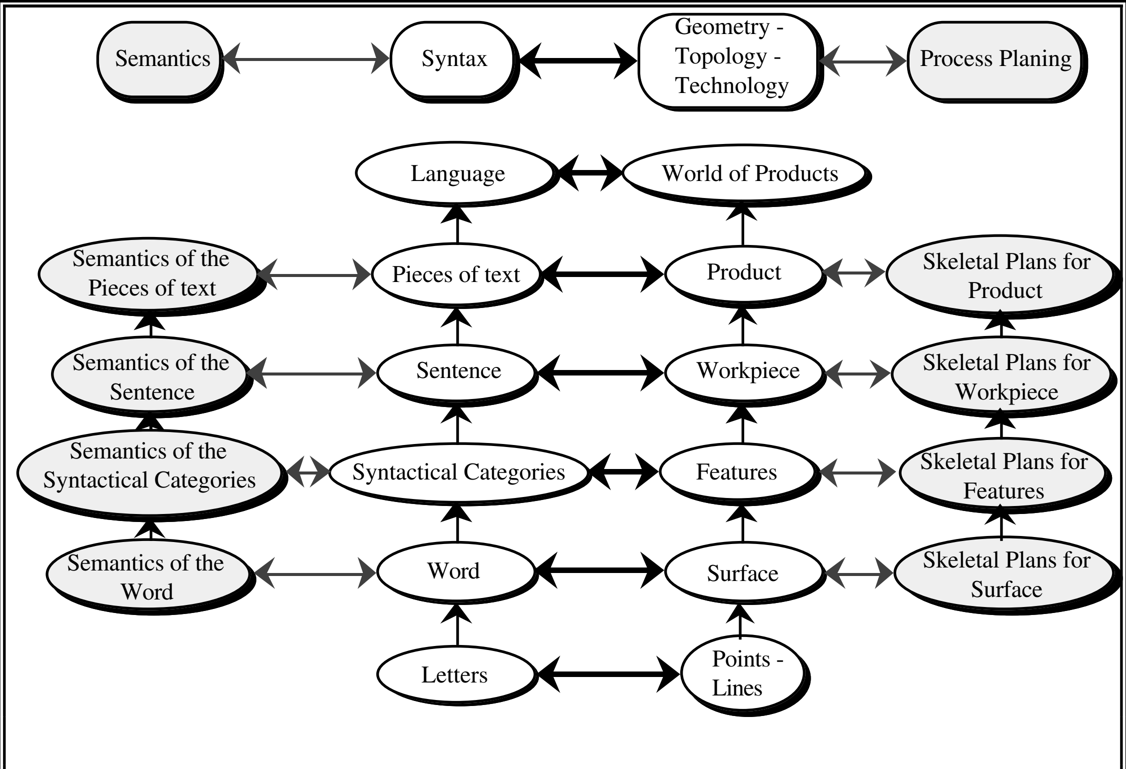


fig. 6 The Language Analogue with Manufacturing Features

manufacturing features. Figure 6 illustrates the analogue between the manufacturing features and a formal language with semantics For a similar natural language analogue cf. [36], p. 63. More informations about the semantics in design can be found in e.g. [37 & 41].

The manufacturer associates *skeletal plans* and also costs with his features. We define a *skeletal plan* as an abstract (part of a) working plan. A machine-ready working plan describes the complete process necessary for the production of a given workpiece in sufficiently detail to be carried out by a machine. A skeletal plan on the other hand describes parts of the whole for producing (a part of) the workpiece on different levels of abstraction. This definition is similar to the one in [24]. The analogue of formal languages with the descriptions of manufacturing features results in our CAPP-system PIM (figure 1).

6. The Feature Representation Language

In this section the representation language FEAT-REP (FEATure-REPresentation) will be presented which allows to represent the feature-language of different experts for use in e.g. feature-based CAPP systems like PIM. Figures 3, 4 and 9 illustrate some requests to FEAT-REP via some characteristics of feature descriptions:

- The first is *feature interaction*. Two or more different features with equal rights, which can be used together to describe a feature, like left and right shoulder, may share some mutual (identical) features, like long turning surface.
- The second is that the same geometrical structures may have *different names*, e.g. groove and insertion. This results from the semantics; the expert divides the semantics of the same geometrical structure into different semantical groups via different feature-names.
- A third characteristic of feature descriptions not yet illustrated is their *contextsensitivity*, e.g. a long turning surface is called a groove ground dependent of the features around it.
- The forth characteristic of feature descriptions is the *fragmentary description* : features could be described via not directly adjoint surfaces respectively features. This maybe the result e.g. of special tools which manufacture not directly adjoint surfaces.
- Finally a characteristic of the feature descriptions is the abstract description level. To describe a feature an expert uses only less geometrical and technological informations; he uses a qualitative description. Quantitative informations are used only when they are needed.

FEAT-REP allows to represent all these characteristics adequate.

6.1. Attributed Node-Label-Controlled Graph Grammars (ANLCGG's)

Before the syntax of FEAT-REP will be shown in the next section we briefly define as theoretical background of our FEAT-REP an attributed node-label-controlled graph grammar (ANLCGG). Introduction and survey can be found in more detail e.g. in [17, 39, 26].

In our paper the term *(feature-)graph* means an attributed finite undirected node labeled topology graph, in the sequel shortly called graph. Such a graph g is formaly given as a 4-tupel FG:= (V, E, Σ, φ), with:

$V :=$ a finite (nonempty) set of *attributed nodes*,

$E := \{(x, y) | x, y \in V$, x is directly <u>topological</u> connected to y$\} \subseteq V \times V$, a finite set of *edges*,

$\Sigma :=$ {names of TEC-REP}$\cup${names of FEAT-REP}, a finite (nonempty) alphabet of node labels or *node sorts* and

$\varphi : V \to \Sigma :=$ a labeling function respectively a *sort function*.

For $v \in V$, $\varphi(v)$ is the *sort of* v. v together with $\varphi(v)$ forms an entity of TEC-REP or FEAT-REP. The class of all graphs with the alphabet of node sorts of Σ is denoted by G_Σ. For a graph $g = (V, E, \Sigma, \varphi)$ the unlabeled graph $g' := (V, E)$, which results from g by eleminating the node labels, is called the *underlying graph of* g and denoted by $g' := unl(g)$.

An *attributed node-label-controlled (feature-)graph grammar* (ANLCGG) is a 4-tuple FGG := (T, N, P, S), with:

T:= {entities of TEC-REP}, a finite (nonempty) set of *terminals*,

N:= {entities of FEAT-REP}, a finite (nonempty) set of *non-terminals*,

P:= a finite set of productions and

$S \in N$ is a node, called the *start node*.

We assume $T \cap N = \emptyset$ and $T \cup N = V$. Note that a featuregraph over T describes a workpiece. A production $p \in P$ is a 4-tuple $p := (l, r, \varepsilon, c)$ where l is a (nonempty) graph over $T \cup N$, called the *left hand side* and r is a (nonempty) graph over $T \cup N$, called the *right hand side*. $p \in P$ is called *contextfree* if $l \in N$, else p is called *contextsensitive*. Note that every production $p \in P$ defines an entity of FEAT-REP, say a feature. ε is an *embedding specification* which determines how the left hand side graph will be joined to the intermediate graph. c is a finite set of *constraints* over l and r, the so-called local dependency relations.

A production $p = (l, r, \varepsilon, c) \in P$ is applied to a featuregraph g by

- searching for a subgraph r' of g with
  - $unl(r') = unl(r)$,
  - for every isomorphic nodes v' of r' and v of r $\varphi(v') = \varphi(v)$ and
  - the set of constraints c is solvable,
- removing r' (and all adjacent edges) from g (leaving the intermediate graph $g \setminus r'$),
- adding l', an isomorphic copy of l disjoint from g, and finally

- adding the embedding edges between l' and g \ r' specified by ε, resulting in a new graph g'.

g *directly concretely derives* g' by replacing graph r' with l' using p, denoted by g $\Rightarrow_p$ g'. Note that the application of a production p to a featuregraph g result in a "shrinking" of g to g': The graph r' describing the feature L' (L' a node of l') is shrinked to the node L'. Every adjacent edge to r' is then adjacent to L'. One key feature of ANLCGG's is that both, the rewriting of a subgraph and the embedding of a newly introduced subgraph are controlled by node sorts.

The TEC-REP entitiy *Cylinder Jacket (CJ)* serves as an example of an attributed node in figure 7. The attributes and their values are attached via a DAG (Directed Acyclic Graph) to the node labeled *Cylinder Jacket*.

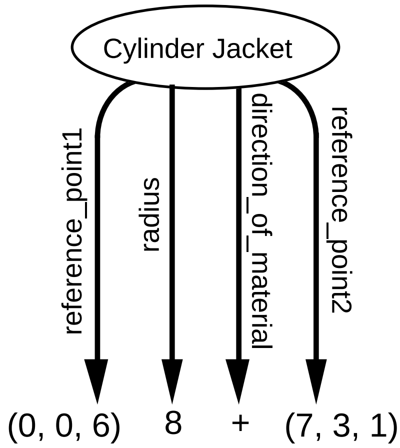
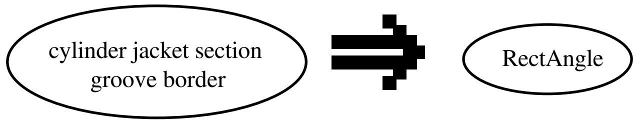


fig. 7 an attributed node in an ANLCGG

In figure 8 examples of ALNCGG rules including relations between the attributes are shown . Figure 8a illustrate a rule where only informations are given to the recognized feature; in figure 8b also constraints are illustrated. The equations of the rules are solved e.g. via unification: the attached DAG's are *compared* according to the type of the equation (e.g. = or >) Note that variables are only bound when equations (=) are used. So attributes can be used to:

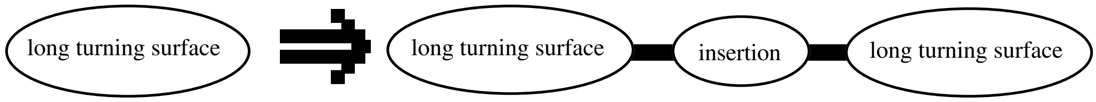
- Information Transport: via unification of atributes of the mother node informations of the daughter nodes
- Information generation: ...

It should be pointed out that features with considerations of dimensions, directions, relative positioning of geometric primitives or any other geometrical or technolical constraints could be defined and recognized via the described ANLCGG's.



cylinder jacket section groove border.geometry = RectAngle.geometry

fig. 8a an ANLCGG rule



long turning surface0.radius = long turning surface1.radius
long turning surface2.radius = long turning surface1.radius

fig. 8b an ANLCGG rule with constraints

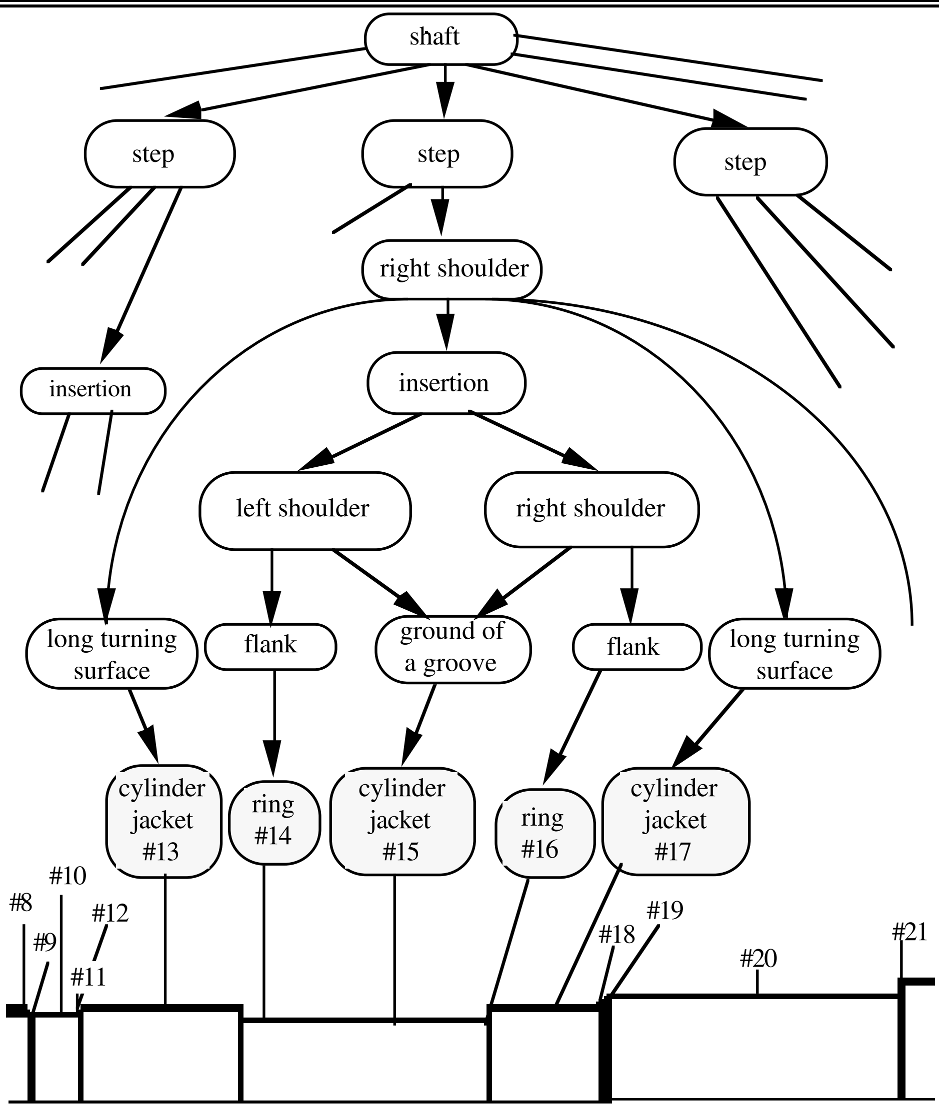


fig. 9 Description of a shaft in terms of an experts features

6.2. The Syntax of FEAT-REP

Now the syntax of FEAT-REP is shown. In this (formal) language a knowledge engineer can represent the experts knowledge about the structure hierachies and manufacturing qualities of workpieces. The characteristics of the features (feature interaction, different names, contextsensitivity, fragmentary description and abstract description) could be represented adequately.

What can be used as the quantitative level of FEAT-REP ? The *Boundary Representation* (*B-Rep*) serves as a basis for the most feature representations in the feature-based systems, i. e. the boundary surfaces of a workpiece are the atomic geometrical entities which are used to describe features. There are also efforts in research to use *Constructive Solid Geometry* (*CSG*) as the atomic geometrical entities (cf. [25] or [48]). In this paper TEC-REP serves as basis of FEAT-REP which is based on the B-Rep. Examples of the TEC-REP entities are shown in figure 2. FEAT-REP itself is a frame like language which is illustrated below.

Feature →
Qualitative_Feature | *Functional_Feature* | *Geometrical_Feature*

Qualitative_Feature →

Featurename:	*featurename*
Featuretype:	*featuretype*
{**Specialize_Feature:**	*featurename*}
{**Subsumes_Features:**	(*list_of_featurenames*)}
{**is_part_of:**	(*list_of_featurenames*)}
{**has_parts:**	(*list_of_featurenames*)}
{**Feature_Rule:**	(*set_of_feature_graph_grammar_rules*)}
{**Rule_Attributes:**	(*list_of_rule_attributes*)}
{**Embedding_Specifications:**	(*list_of_embedding_specifications*)}
{**Described_Feature:**	*featurename*}
Description:	(*list_of_qualitative_constraints*)
{**Feature_Context:**	(*list_of_context_constraints*)}

Functional_Feature →

Featurename:	*featurename*
Featuretype:	*featuretype*
{**Specialize_Feature:**	*featurename*}
{**Subsumes_Features:**	(*list_of_featurenames*)}
{**is_part_of:**	(*list_of_featurenames*)}
{**has_parts:**	(*list_of_featurenames*)}
{**Feature_Rule:**	(*set_of_feature_graph_grammar_rules*)}
{**Rule_Attributes:**	(*list_of_rule_attributes*)}
{**Embedding_Specifications:**	(*list_of_embedding_specifications*)}
{**Described_Feature:**	*featurename*}
Description:	(*list_of_functional_constraints*)
{**Feature_Context:**	(*list_of_context_constraints*)}

Geometrical_Feature →	
Featurename:	*featurename*
{**Specialize_Feature:**	*featurename*}
{**Subsumes_Features:**	(*list_of_featurenames*)}
{**is_part_of:**	(*list_of_featurenames*)}
has_parts:	(*list_of_featurenames*)
Feature_Rule:	(*set_of_feature_graph_grammar_rules*)
Rule_Attributes:	(*list_of_rule_attributes*)
{**Embedding_Specifications:**	(*list_of_embedding_specifications*)}
{**Feature_Context:**	(*list_of_context_constraints*)}

Featurename and Featuretype together identify the feature. The featurenames are given by the expert; the featuretypes are the differentations in the definition of the term feature, like *geometrical drilling* feature or *functional design* feature. Via Specialize_Feature and Subsumes_Features a hierachical structure over the features is constructed. This structure is generated and managed by a KL-ONE like conceptual language formalism called TAXON [7]. Is_part_of and Has_parts makes the part-of relation explicit. It is a redundant information like Subsumes_Features, too, and helps to make it easier to read the feature descriptions. Feature_rule is a set of alternative graph grammar rules which describes the featuregraph. Via this rule the parts of a feature are set into a (topological) relation. The attributes of a rule are divided via Rule_Attributes and Feature_Context into the attributes which depend only on the data of the featureparts itself and the attributes which depend on the data of the feature context. The context also includes informations about machines ore tools. When Embedding_Specifications is not specified, the default specification is used: Every adjacent edge to the right hand side of the rule is adjacent to the left hand side of the rule. Described_Feature is the link to the geometrical features but the underlaying geometry can also be described explicitly. Description is the list of the functional or qualitative constraints which describe the feature.

context_constraint →
predicate

functional_constraint →
predicate

qualitative_constraint →
predicate

rule_attribute →
geometrical_equation |
technological_equation |
tolerance_equation

The constraints and attributes (relations between attributes) are just described via predicates, where the attributes are restricted to a given set of equations. They could both be handled by the constraint system CONTAX or/and FIDO [1]. Note that variables will only be bound when equations (predicates) of type "=" are used; predicates and equations of other type over unbound variables will always be failure.

equation_name →
    < | > | ≥ | ≤ | =

geometrical_equation→
    (*equation_name geometrical_attribute geometrical_attribute*)|
    (*equation_name geometrical_attribute value*)|

predicate→
    (*predicate_name list_of_terms*)

predicate_name→
    < | > | ≥ | ≤ | = | useable | solid | <system or user defined predicate names> ...

technological_equation→
    (*equation_name technological_attribute technological_attribute*)|
    (*equation_name technological_attribute value*)|

tolerance_equation→
    (*equation_name tolerance_attribute tolerance_attribute*)|
    (*equation_name tolerance_attribute value*)

With these predicates relations between the attributes and relations between an attribute and a constant could be described. They can be used with different functions:

- First they can be used as comparison of values (of daughter nodes), e.g. the comparison of dimensions;
- Second they can be used to inherit informations from the daughter nodes to the mother node; e.g. the boundary points of the daughter nodes;
- Third they can be used to fill in new informations to the attributes of the mother node by means of functions, e.g. to compute the maxium length;
- Finally they can be used to compare constants with values of daughter nodes, e.g. the surface finish of a daughter node with a given restriction to the mother node.

So in conclusion the predicates can be used to compare attributes with attributes or constants and they could be used to pass or generate informations.

geometrical_attribute →
({(reference_point1 *range_of_values*)}
{(reference_point2 *range_of_values*)}
{(radius1 *range_of_values*)}
{(radius2 *range_of_values*)}
...
{(<Attributes of TEC-REP> *range_of_values*)}
{(direction_of_material *range_of_values*)})

technological_attribute →
({(surface_finish *range_of_values*)}
{(hardness *range_of_values*)}
...
{(<Attributes of TEC-REP> *range_of_values*)}
{(value *range_of_values*)})

tolerance_attribute →
({(nominal_size *range_of_values*)}
{(min_size *range_of_values*)}
{(max_size *range_of_values*)}
...
{(<Attributes of TEC-REP> *range_of_values*)}
{(tolerance_extent *range_of_values*)})

As attributes all attributes of the TEC-REP entities are used.

edge →
(*parameter-definition parameter-definition*) | (*featurename featurename*)
| (*parameter-definition featurename*)

embedding_specification →
(*list_of_edges*)

feature_graph →
(*list_of_edges*)

feature_graph_grammar_rule →
(*left_hand_side_graph right_hand_side_graph*)

left_hand_side_graph →
feature_graph

right_hand_side_graph →
feature_graph

The graph grammar rules are productions of a formal language where the left hand side and the right hand side are graphs. In every rule only one nonterminal will be rewritten, even though on the right hand side and on the left hand side nonterminals could occcur as context.

digit →
digit digit | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

featureapplication →
"design" | "assembling" | "milling" | "drilling" | "turning" | ...

featurekind →
"atomical" | "geometrical" | "functional" | "qualitative"

featurename →
string

featuretype →
featurekind featureapplication

function →
(*function_name list_of_terms*)

function_name →
CAR | CDR | + | / | <system or user defined function names> ...

letter →
a | b | c | . | A | B | C | ...

list_of_context_constraints →
(*context_constraint**)

list_of_edges →
(*edge**)

list_of_embedding_specifications →
(*embedding_specification**)

list_of_featurenames →
(*featurename**)

list_of_functional_constraints →
(*functional_constraint**)

list_of_params →
(*param**)

list_of_qualitative_constraints →
(*qualitative_constraint**)

list_of_rule_attributes →
(*rule_attributet**)

list_of_terms→
(*term**)

lower_value→
value

number→
sign digit | *sign digit* , *digit* | *sign digit* , *digit sign* **E** *digit*

parameter→
x | y | z | ...

parameter-definition→
(*parameter range_of_values*)

range_of_values→
(*lower_value upper_value*) | *value*

set_of_feature_graph_grammar_rules →
(*feature_graph_grammar_rule**)

sign →
+ | -

string →
string symbol | *letter*

symbol →
letter | ! | " | ? | - | _ | / | ...

term →
range_of_values | *function* | *parameter-definition*

upper_value→
value

value→
string | *number*

These specifications describe the needed terms like strings, terms ore numbers. An example of the turning feature insertion is given below and illustrated in figure 9.

Featurename:	*insertion*
Featuretype:	*geometrical turning feature*
Subsumes_Features:	**(** *O-ring_groove* **)**
is_part_of: **(** *left_shoulder, right_shoulder, long_turning_surface*, *step* **)**	
has_parts:	**(** *left_shoulder, right_shoulder* **)**
Feature_Rule:	**(** *(((insertion), (left_shoulder, right_shoulder)), ((insertion), (left_shoulder, insertion, right_shoulder)))* **)**
Rule_Attributes:	**(** *nil* **)**

7. Feature Recognition

As proposed in the previous sections the importance of feature recognition in manufacturing stems from the fact that each feature can be associated with knowledge about how the feature should be manufactured; this information can be used to generate a process plan. From this point of view feature recognition forms a major component of the CAD/CAM interface for CAPP. In our paper we concentrated on the recognition of geometrical and qualitative features; the functional features are important for design only. Working with manufacturing features means to recognize these features from the CAD data to generate a working plan. Working with design features means to construct by means of these features and to expand them to the CAD data. The most recent developments in this research field can be read in e.g. [15, 6, 5, 22, 29] and [16].

Within our current research the features will be recognized or expanded by parsing methods which are based on graph matching methods and heuristics (background knowledge). This is facilitated through the representation of the feature definitions in a well-formed attributed node-label-controlled graph grammar. The *feature-parser* finds the complete set of features derivable from the productions of an ANLCGG given the grammar and a workpiece described in the terms of TEC-REP (an augmented topology graph representing the geometry and technology of a workpiece). So the problem of feature recognition is the problem of finding isomorphic subgraphs, in general a NP-complete problem [4]. But it maybe come solvable in $O(n^x)$ time using e.g. the method described in [35]: "*The technique is, to incorporate <u>application dependent knowledge</u> systematically ...*". The detail of the feature recognition algorithm will be published in a separate paper. Besides these activities there are examinations to recognize features via combined logical forward and backward reasoning in conjunction with taxonomies [32].

One problem that arises in the CAPP systems from the integration of CAD and CAM is that a workpiece must be transformed through different feature-languages, e.g. the one of a designer, a driller or a turner. On this way the workpiece passes different qualitative description languages. The gap between the single qualitative levels will be brigded by a quantitative description level, e.g. TEC-REP. This level contains all information needed to generate another qualitative description of the workpiece. But why forget the previous qualitative description? So when another qualitative description of the workpiece will be generated, the previous qualitative description could be used to make this generation more efficient. In figure 10 this method is illustrated.

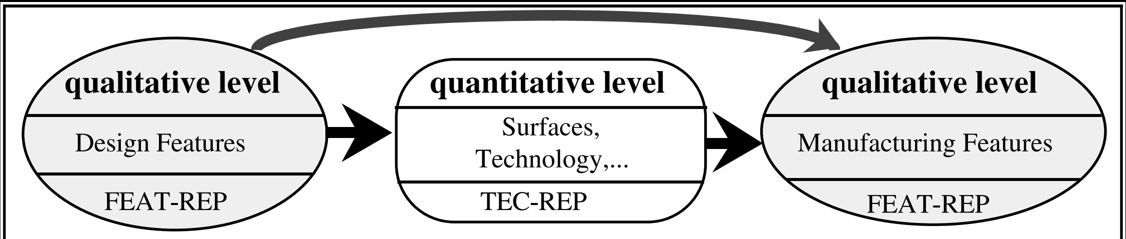


fig. 10 Getting from one qualitative description to the other

For example when a designer constructs a Seeger circlip ring groove the same geometry can be seen as groove by the manufacturer; only the feature-names and the semantics must be changed. This method will be integrated in the feature recognition algorithm.

8. Conclusion

Grammars are the rewriting systems that define languages in terms of syntax, semantics and pragmatics. The relationship between grammars and languages is that a grammar strictly defines an associated language. In our paper we show that it is possible to describe features by means of formal languages via attributed node-label-contolled graph grammars. The area of formal languages is a well established field of research and provides a powerful set of methods like parsing and knowledge about problems, their complexity and how they could be solved efficiently. The use of formal languages for feature descriptions facilitates the application of these results to the area of feature recognition and CAPP. As result ANLCGG's enables a user to define his own feature-language containing complex features and makes feature recognition a parsing process for workpiece interpretation.

The graph grammar based formalism FEAT-REP is a powerful and general tool to represent feature descriptions. A feature language defined in this fomalism represents a link between the quantitative (low-level) geometrical/technological representation and the qualitative (high-level) abstractions, as qualitative entities are expressed in terms of quantitative ones. Because the quantitative description of a workpiece can be seen as a topological graph, the features can be recognized by graph-based parsing.

In future research a domain dependent graph-based parsing algorithm based on ANLCGG's will be developed. Currently, a small feature-grammar of one of our experts has been implemented using the extended D-PATR system (Karttunen L.: *D-PATR: A Development Environment for Unification-Based Grammars*, CSLI Report, CSLI-86-68), a formalism to represent unification-based grammars. Our quantitative representation formalism TEC-REP serves as a lexicon in this system.

9. References

[1] *CONTAX: Constraint-Based Reasoning over Taxonomies*, forthcoming 1991.

[2] Abecker, A. and Hanschke, P.: *TAXON: Instruction for use*, october 1990.

[3] Abeln, O.: *Die Ca-Techniken in der industriellen PraxisHandbuch der computergestützten Ingenieur-Methoden,* Hanser Verlag (1990).

[4] Aho, A.V., Hopcroft, J.E., and Ullman, J.D.: *The Design and Analysis of Computer Algorithms,* Addison-Wesley (1974).

[5] Anderson, D.C. and Henderson, M.R.: *Computer Recognition and Extraction of Form Features: A CAD/CAM Link*. Computers in Industry (6) 4 (1984), 315-325.

[6] Anderson, D.C., Chang, T.C., and Mitchell, O.R.: QTC- An Integrated Design/Manufacturing/Inspection System for Prismatic Parts. In *International Computers in Engineering Conference and Exhibition,* july/august 1988, pp. 417-426.

[7] Baader, F. and Hanschke, P.: A Scheme for Integrating Concrete Domains into Concept Languages. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence,* 1991.

[8] Bernardi, A., Klauck, C., and Legleitner, R.: *STEP: Überblick über eine zukünftige Schnittstelle zum Produktdatenaustausch.* Dokument, D-90-04 Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, Postfach 20 80, D-6750 Kaiserslautern, september, 1990.

[9] Bernardi, A., Klauck, C., and Legleitner, R.: *Abschlußbericht des Arbeitspaketes PROD.* Dokument, D-90-03 Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, Postfach 20 80, D-6750 Kaiserslautern, september, 1990.

[10] Bernardi, A., Klauck, C., and Legleitner, R.: *Formalismus zur Repräsentation von Geometrie- und Technologieinformationen als Teil eines Wissensbasierten Produktmodells.* Dokument, D-90-05 Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, Postfach 20 80, D-6750 Kaiserslautern, december, 1990.

[11] Bernardi, A., Klauck, C., and Legleitner, R.: PIM: Planning in Manufacturing. In *forthcoming,* DFKIGmbH, 1991.

[12] Bernardi, A., Klauck, C., and Legleitner, R.: *TEC-REP: Repräsentation von Geometrie- und Technologieinformationen.* Dokument, D-91-07 Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, Postfach 20 80, D-6750 Kaiserslautern, june, 1991.

[13] Bernardi, A., Boley, H., Hanschke, P., Hinkelmann, K., Klauck, C., Kühn, O., Legleitner, R., Meyer, M., Richter, M.M., Schmalhofer, F., Schmidt, G., and Sommer, : ARC-TEC: Acquisition, Representation and Compilation of Technical Knowledge. In *Expert Sytems and their Applications:Tools, techniques and Methods,* 1991, pp. 133-145.

[14] Chang, G.J. and Henderson, M.R.: FRAPP: Automated Feature Recognition and Process Planning from Solid Model Data. In *International Computers in Engineering Conference and Exhibition,* july/august 1988.

[15] Chang, T.C.: *Expert Process Planning for Manufacturing,* Addison-Wesley (1990).

[16] Chuang, S.H. and Henderson, M.R.: *Compound Feature Recognition by Web Grammar Parsing*. Research in Engineering Design 2 (1991), 147-158.

[17] Claus, V., Ehrig, H., and Rozenberg, G. (Eds). *Graph-Grammars and Their Application to Computer Science and Biology*. 1978.

[18] Cunningham, J.J. and Dixon, J.R.: Designing with features: the origin of features. In *International Computers in Engineering Conference and Exhibition,* july/august 1988, pp. 237-243.

[19] Cutkosky, M.R. and Tenenbaum, J.M.: Features in Process-Based Design. In *International Computers in Engineering Conference and Exhibition,* july/august 1988, pp. 557-562.

[20] Dixon, J.R. and Finger, S.: *A Review of Research in Mechanical Engineering Design. Part II :Representations, Analysis, and Design for the Life Cycle*. Engineering DesignSpringer-Verlag New York Inc. (1) 2 (1989), 121-137.

[21] Dixon, J.R. and Finger, S.: *A Review of Research in Mechanical Engineering Design. Part I : Desriptive, Prescriptive, and Computer-Based Models of Design Processes*. Engineering DesignSpringer-Verlag New York Inc. (1) 2 (1989), 51-67.

[22] Finger, S.: Parsing Features in Solid Geometric Models. In *ECAI90,* 1990.

[23] Finger, S., Fox, M.S., Prinz, F.B., and Rinderle, J.R.: *Concurrent Design*. Applied Artificial Intelligence Special Issue and AI in Manufacturing (1990).

[24] Friedland, P.E. and Iwasaki, Y.: *The Concept and Implementation of Skeletal Plans*. Journal of Automated Reasoning (1)(october 1985), 161-208.

[25] Fu, K.S. and Lee, Y.C.: *Machine Understanding of CSG: Extraction and Unification of Manufacturing Features*. IEEE Computer Graphics and Applications (7) 1 (january 1987), 20-32.

[26] Goos, G. and Hartmans, J. (Eds). *Graph-Grammars and Their Application to Computer Science*. H. Ehrig and M. Nagl and G. Rozenberg and A. Rosenfeld, december 1986.

[27] Gossard, D.C. and Hirschtick, J.K.: Geometric Reasoning for Design Advisory Systems. In *International Computers in Engineering Conference and Exhibition,* july 1986, pp. 263-270.

[28] Gossard, D.C. and Sakurai, H.: Shape Feature Recognition from 3D Solid Models. In *International Computers in Engineering Conference and Exhibition,* july/august 1988, pp. 515- 519.

[29] Gossard, D.C. and Sakurai, H.: *Recognizing Shape Features in Solid Models*. IEEE Computer Graphics and Applications (1990), 22-32.

[30] Grätz, J.: *Handbuch der 3D - CAD - TechnikModellierung mit 3D-Volumensystemen,* Siemens - Aktiengesellschaft (1989).

[31] Heragu, S.S. and Kusiak, A.: *Analysis of Expert Systems in Manufacturing Design*. IEEE Transactions on Systems, Man, and Cybernetics (SMC-17) 6 (november 1987), 898-912.

[32] Hinkelmann, K.: Combining Forward and Backward Logic Evaluation for Feature Recognition. In *forthcoming,* 1991.

[33] Janssens, D. and Rozenberg, G.: A Survey of NLC Grammars. In *Trees in Algebra and Programming8th Colloquium,* march 1983, pp. 114-128.

[34] Joshi, S.B.: *CAD interface for automated process planning*, Ph.D. dissertation, Purdue University, august 1987.

[35] Kaul, M.: Practical Applications of Precedence Graph Grammars. In *Graph-Grammars and Their Application to Computer Science,* Goos, G. and Hartmans, J., H. Ehrig and M. Nagl and G. Rozenberg and A. Rosenfeld, december 1986, pp. 326-342.

[36] Kyprianou, L.K.: *Shape Classification in Computer-Aided Design*, Ph.D. dissertation, Christ`s College, University of Cambridge, P.O Box 1745NicosiaCyprus, July 1980.

[37] Mullins, S. and Rinderle, J.R.: *Grammatical Approaches to Engineering Design, Part I: An Introduction and Commentary*. Research in Engineering Design 2 (1991), 121-135.

[38] Murakami, T. and Nakajima, N.: Design-Diagnosis using Feature Description. In *IFIP WG 5.2 Workshop on Intelligent CAD Systems,* Gossard, D., IFIP, october 1987, pp. 169-185.

[39] Nagl, M.: *Graph-Grammatiken: Theorie, Implementierung, Anwendungen,* Friedrich Vieweg & Sohn (1979).

[40] Pratt, M.J.: *Solid Modeling and the Interface Between Design and Manufacture*. IEEE Computer Graphics and Applications (july 1984), 52-59.

[41] Rinderle, J.R.: *Grammatical Approaches to Engineering Design, Part II: Melding Configuration and Parametric Design Using Attribute Grammars*. Research in Engineering Design 2 (1991), 137-146.

[42] Rogers, M.T. and Shah, J.J.: *Expert form feature modelling shell*. Computer Aided Design (20) 9 (november 1988), 515-524.

[43] Rozenberg, G.: An Introduction to the NLC Way of Rewriting Graphs. In *Graph-Grammars and Their Application to Computer Science,* Goos, G. and Hartmans, J., H. Ehrig and M. Nagl and G. Rozenberg and A. Rosenfeld, december 1986, pp. 55-66.

[44] Schuette, A.: *Knotenattributierte Kontextfreie Graphgrammatiken.* Bericht, 1/1984 Erziehungswissenschaftliche Hochschule Rheinland-Pfalz (EWH), Rheiau 3-4, D-5400 Koblenz, 1984.

[45] Schuette, A.: *Einführung in Theorie und Konzepte von attributierten Zeichenketten- und Graphgrammatiken.* Bericht, 1/1985 Erziehungswissenschaftliche Hochschule Rheinland-Pfalz (EWH), Rheiau 3-4, D-5400 Koblenz, 1985.

[46] Schuster, R.: *Graphengrammatiken und Grapheneinbettungen: Algorithmen und Komplexität*, Ph.D. dissertation, Universität Passau, Fakultät für Informatik, Universität Passau, Fakultät InformatikPostfach 2540, 8390 PassauMIP-8711, june 1987.

[47] Tengvald, E.: *The Design of Expert Planning Systems: An Experimental Operations Planning System for Turning*, Ph.D. dissertation, Lingköping University, Department of Computer and Information ScienceLinköping UniversityS-58183 Linköping, Sweden, 1984.

[48] Woodwark, J.R.: *Some speculations on feature recognition*. Computer Aided Design (20) 4 (may 1988), 189-196.

10. Appendix

The listet paper will be published in the Proceedings, and presented in the IV International Symposium on Artificial Intelligence: Applications in Informatics, to be heldin Cancún, México on November 13-15 1991.

FEAT-REP:
Representing Features in CAD/CAM

C. Klauck, A. Bernardi, R. Legleitner

ARC-TEC Project
German Research Center for Artificial Intelligence
DFKI GmbH, Postfach 2080, D-6750 Kaiserslautern, Germany
Telefon: +49631/205-3477, 205-4068
Fax: +49631/205-3210
email: {klauck|bernardi|legleit}@dfki.uni-kl.de

Abstract

When CAD/CAM experts view a workpiece, they perceive it in terms of their own expertise. These terms, called *features*, which are build upon a *syntax* (geometry) and a *semantics* (e.g. skeletal plans in manufacturing or functional relations in design), provide an abstraction mechanism to facilitate the creation, manufacturing and analysis of workpieces. Our goal is to enable experts to represent their own *feature-language* via a *feature-grammar* in the computer to build *feature-based* systems e.g. CAPP systems. The application of formal language terminology to the feature definitions facilitates the use of well-known formal language methods like parsing in conjunction with our flexible knowledge representation formalism FEAT-REP.

Keywords: feature, feature recognition, feature-language, feature-grammar, Attributed Node-Label-Controlled Graph Grammars

1. Introduction

An important step towards truely Computer Integrated Manufacturing (CIM) is the Computer Aided Process Planning (CAPP). A CAPP system will use the information provided by CAD (Computer Aided Design) to generate the process plan for the manufacturing of the workpiece by means of CAM (Computer Aided Manufacturing).

The solid modellers currently used in CAD describe a workpiece only in terms of lower-level entities like faces, edges, vertices (topology), surfaces, lines and points (geometry), or volumetric primitives like cylinders or cones. While these lower-level entities represent the complete quantitative information about a workpiece, efficient planning strategies rely on higher-level (qualitative) information supporting abstract reasoning to accomplish their goals. In our approach these higher-level entities are the so-called *features* which must be extracted from the data of the CAD models [7, 5]. In the discussion about the role of solid modelling as the interface between design and manufacturing these higher-level informations build the bridge between the workpiece created by the designer and the process plan. Employing features, an experts knowledge in this domain can be suitable formalized and used in planning systems ([3]).

The proposed system PIM (Planning In Manufacturing) in [3] recognizes features in a given representation of a workpiece, finds skeletal plans associated to these features, and refines these plans to the CLDATA code (Cutter Location DATA) necessary for

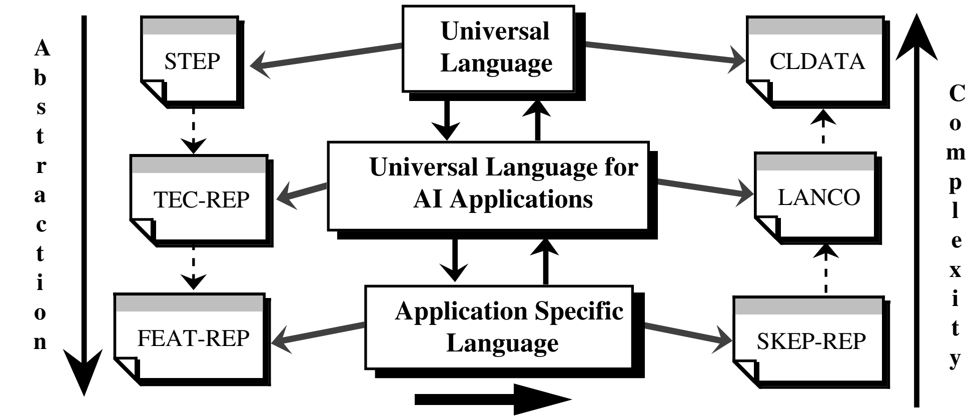


fig. 1 The basic method of PIM

manufacturing. This sequence of abstractions/refinements is illustrated in figure 1 and follows the expertise model of human experts (cf. [13]). To bridge the gap between the geometric description e.g. represented in STEP (STandard for the Exchange of Product model data) and the manufacturing instructions e.g. represented in CLDATA code, the sequence of representations on different abstraction levels reduces this problem to the problem of finding an associated *skeletal plan* to a given workpiece represented in terms of features. So representing and recognizing features is a necessary step to bridge the gap between CAD and CAM. It is important to note that in general different domains like design, turning or milling leads to different features and that a standardization of all features is just unreasonable.

In this paper we show that it is possible to describe features by means of formal languages via attributed node-label-contolled graph grammars. The area of formal languages is a well established field of research and provides a powerful set of methods like parsing and knowledge about problems, their complexity and how they could be solved efficiently. The use of formal languages for feature descriptions facilitates the application of these results to the area of feature recognition and CAPP.

2. What are Features ?

Currently there is no consensus on a precise definition of the term *feature*. Most researchers working in this area agree that a feature is an abstraction of lower-level design and manufacturing information which depends on the context of the machine shop. Features that are required for design may differ considerably from those required for manufacturing or assembly, even though they may be based on the same lower-level entities. Cunsulting several experts of manufacturing and design showed that these differences are reasonable.

John R. Dixon and John J. Cunningham have defined a feature as "*any geometric form or entity that is used in reasoning in one or more design or manufacturing activities*"[6]. Tien-Chien Chang has defined a feature in his book [5] as "*a subset of geometry on an engineering part which has a special design or manufacturing characteristic.*". Other similar definitions of features can be found in e.g. [7].

We define the term *feature* as a description element based on geometrical and technological data of a product which an expert in a domain associates with certain informations. They are firstly distinguished by their kind as

- functional features, e.g. *seat of the rolling bearing* or *O-ring groove*,
- qualitative features, e.g. *bars* or *solid workpiece*,
- geometrical (form-) features, e.g. *shoulder*, *groove* or *drilled hole*,
- atomic features, e.g. *toroidal shell*, *ring*, *shape tolerance* or *surface finish*.

and they are secondly distinguished by their application as

- design features, e.g. *crank* or *coupler*,
- manufacturing features:
  - turning features, e.g. *shoulder* or *neck*,
  - milling features, e.g. *step* or *pocket*,
  - drilling features e.g. *stepped-hole* or *lowering*,
  - ...
- ...

Our definition follows the one of Tien-Chien Chang [5] and is distinguished by the emphasis on an expert in a domain. In particular every feature will be defined by a respective expert because his area, like machines, tools or their characteristics, and his ideas, creativity and experience, like special tricks, is included in this definition. In this sense features can been seen as a *language* of an expert in a domain. It is important to note that this language represents the *know-how* of the expert respectively the machine shop and that this language is an individual ("expert in a domain" dependent) one. It is also important to note that such a language has a syntax <u>and</u> a semantics. What we interpret as syntax and semantics of these *feature-languages* will be explained in the next section. So it is incumbent upon the XPS-shells or -tools only to define a representation language for features and not the features itself; they must be defined individually for every XPS in its individual area.

3. Syntax and Sematic of Feature-Languages

In the previous chapter we defined the term feature. There also is mentioned briefly an analogue between the feature descriptions and (formal) languages which results in the term *feature-language*. In this chapter the syntax and semantics of feature-languages will be described in general. Before we will discuss the syntax it should be pointed out that the expert chooses/creates a syntax of the features which dependents on the information associated with the features.

3.1 The Syntax

The first important issue about the features is that the expert bases his definitions on the boundary surfaces and the technological informations of the workpiece, like tolerances or surface finish, which are assigned to one or more surfaces. Our representation formalism TEC-REP ([4]) supplies these entities which are used as atomic features. TEC-REP also supplies a topology graph to represent the neighbourhoodness of surfaces.

To define the geometrical features the expert uses the atomic features and the geometrical features itself. One simple example of features described by an expert is shown in figure 4.

It is important to note that rules for string grammars are only sufficient when features of rotational symmetric parts are described; in general graph-based rules are needed (cf. [12]) because the features will be defined by the topological graph of their parts. An example can be seen in figure 2.

The descriptions of functional features are based upon the descriptions of geometrical features and differ in the connection to other products. The functionality is defined via the description of the functional relation between the functional feature and one or more other products. The syntax rules of these features differ in the additional attributes which describe the functional relation and the technological restrictions. The descriptions of qualitative features are also based upon geometrical features and represent a more abstract description of a workpiece. Their syntax rules differ in the additional attributes which describe the technological and geometrical restrictions. As conclusion we can state out that the geometrical description in addition with attributes about the context, functionality and technology forms the syntax of a feature.

3.2 The Semantics

Now we can describe the semantics of a feature-language. The main thing of the features is, that the expert associates certain informations with the features. From this view the syntax of the features can be seen as a vessel to carry the information, the *semantics* of the features. What kind of informations associates the expert with his features ? This depends on his working field. A designer for example associates first the functionality and the costs with his features. So when he says "seat of the rolling bearing" he first describes the syntax of the feature, e.g. geometry and technology, and secondly he describes the semantics of the feature, e.g. that this part will be used as a seat. Our research concentrates on the semantics of the manufacturing features. Figure 3 illustrates the analogue between the manufacturing features and a formal language with semantics. More information about the semantics in design can be found in e.g. [11 & 12].

The manufacturer associates *skeletal plans* and also costs with his features. We define a *skeletal plan* as an abstract working plan. A machine-ready working plan describes the complete process necessary for the production of a given workpiece in sufficient detail to be carried out by a machine. A skeletal plan on the other hand describes parts of the whole for producing (a part of) the workpiece on different levels of abstraction. This definition is similar to the one in [8]. The analogue of formal languages with the descriptions of manufacturing features results in our CAPP-system PIM (figure 1).

4. The Feature Representation Language

In this section the representation language FEAT-REP (FEATure-REPresentation) will be presented which allows to represent the feature-language of different experts for use in e.g. feature-based CAPP systems like PIM. Figures 2 and 4 illustrate some requests to FEAT-REP via some characteristics of feature descriptions: The first is *feature interaction*. Two or more different features with equal rights, which can be used together to describe a feature, like left and right shoulder, may share some mutual (identical) features, like long turning surface. The second is that the same geometrical structures may have *different names*, e.g. groove and insertion. This results from the semantics; the expert divides the semantics of the same geometrical structure into different semantical groups via different feature-names. A third characteristic of feature descriptions not yet illustrated is their *contextsensitivity*, e.g. a long turning surface is called a groove ground dependent of the features around it. The forth characteristic of feature descriptions is the *fragmentary description* : features could be described via not directly adjoint surfaces respectively features. This may be the result e.g. of special tools which manufacture not directly adjoint surfaces. Finally a characteristic of the feature descriptions is the abstract description level. To describe a feature an expert uses only less geometrical and technological informations; he uses a qualitative description. Quantitative informations are used only when they are

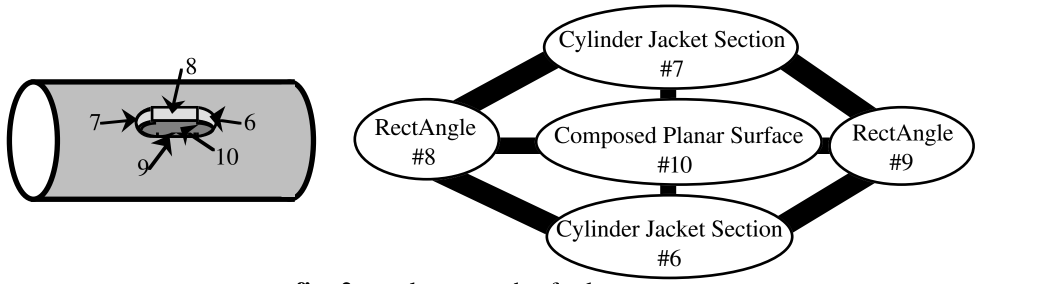


fig. 2 topology graph of a key groove

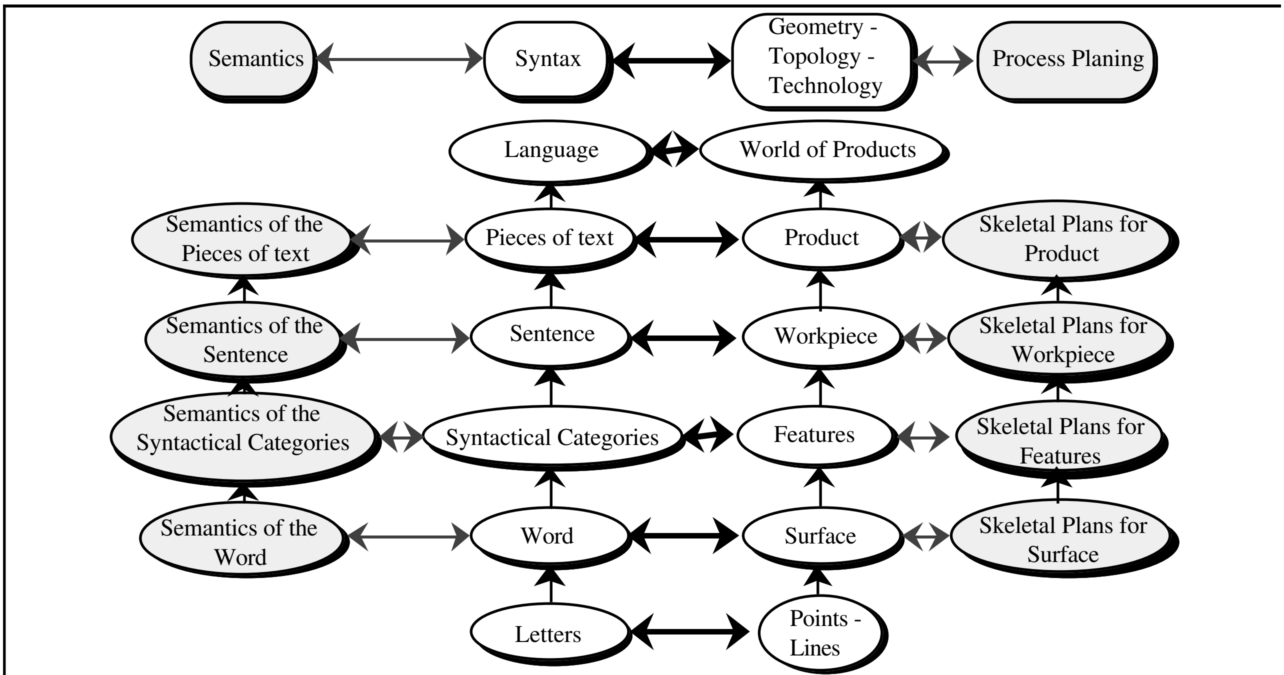


fig. 3 The Language Analogue with Manufacturing Features

needed. FEAT-REP allows to represent all these characteristics adequately.

4.1 Attributed Node-Label-Controlled Graph Grammars (ANLCGG's)

Before the syntax of FEAT-REP will be shown in the next section we briefly define as theoretical background of our FEAT-REP an attributed node-label-controlled graph grammar (ANLCGG). Introduction and survey can be found in more detail e.g. in [9].

In our paper the term *(feature-)graph* means an attributed finite undirected node labeled topology graph, in the sequel shortly called graph. Such a graph g is formaly given as a 4-tupel FG:= (V, E, Σ, φ), with:

V:= a finite (nonempty) set of *attributed nodes*,

E := $\{(x, y) \mid x, y \in V$, x is directly topological connected to y$\} \subseteq V \times V$, a finite set of *edges*,

Σ := {names of TEC-REP}$\cup${names of FEAT-REP}, a finite (nonempty) alphabet of node labels or *node sorts* and

$\varphi : V \rightarrow \Sigma$:= a labeling function respectively a *sort function*.

For $v \in V$, $\varphi(v)$ is the *sort of* v. v together with $\varphi(v)$ forms an entity of TEC-REP or FEAT-REP. The class of all graphs with the alphabet of node sorts of Σ is denoted by G_Σ. For a graph g = (V, E, Σ, φ) the unlabeled graph g' := (V, E), which results from g by eleminating the node labels, is called the *underlying graph of* g and denoted by g' := unl(g).

An attributed node-label-controlled (feature-)graph grammar (ANLCGG) is a 4-tuple FGG := (T, N, P, S), with:

T:= {entities of TEC-REP}, a finite (nonempty) set of *terminals*,

N:= {entities of FEAT-REP}, a finite (nonempty) set of *non-terminals*,

P:= a finite set of productions and

$S \in N$ is a node, called the *start node*.

We assume $T \cap N = \emptyset$ and $T \cup N = V$. Note that a featuregraph over T describes a workpiece. A production $p \in P$ is a 4-tuple p := (l, r, ε, c) where l is a (nonempty) graph over $T \cup N$, called the *left hand side* and r is a (nonempty) graph over $T \cup N$, called the *right hand side*. $p \in P$ is called *contextfree* if $l \in N$, else p is called *contextsensitive*. Note that every production $p \in P$ defines an entity of FEAT-REP, say a feature. ε is an *embedding specification* which determines how the left hand side graph will be joined to the intermediate graph. c is a finite set of *constraints* over l and r, the so-called local dependency relations.

A production p = (l, r, ε, c) $\in$ P is applied to a featuregraph g by

- searching for a subgraph r' of g with
- unl(r') = unl(r),
- for every isomorphic nodes v' of r' and v of r $\varphi(v') = \varphi(v)$ and
- the set of constraints c is solvable,
- removing r' (and all adjacent edges) from g (leaving the intermediate graph g \ r'),
- adding l', an isomorphic copy of l disjoint from g, and finally

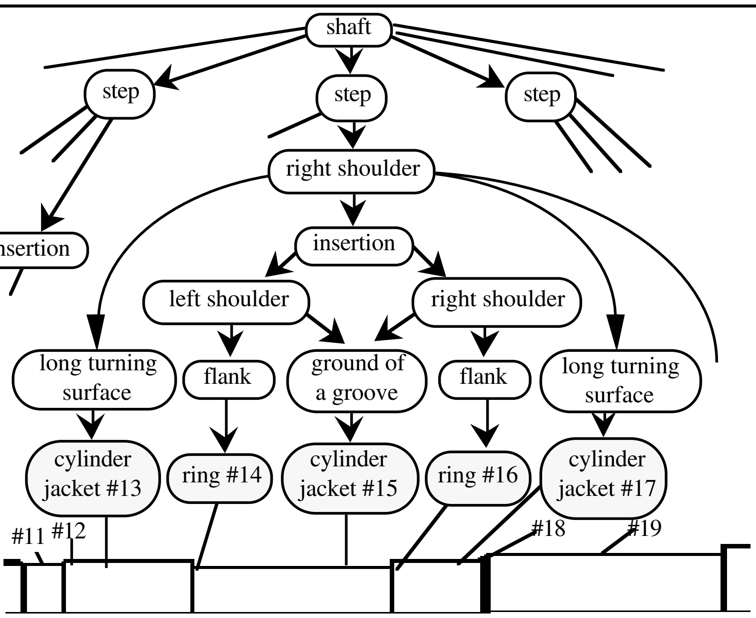


fig. 4 Description of a shaft in terms of an experts features

- adding the embedding edges between l' and g \ r' specified by ε, resulting in a new graph g'.

g *directly concretely derives* g' by replacing graph r' with l' using p, denoted by g $\Rightarrow_p$ g'. Note that the application of a production p to a featuregraph g result in a "shrinking" of g to g': The graph r' describing the feature L' (L' a node of l') is shrinked to the node L'. Every adjacent edge to r' is then adjacent to L'. One key feature of ANLCGG's is that both, the rewriting of a subgraph and the embedding of a newly introduced subgraph are controlled by node sorts.

4.2 The Syntax of FEAT-REP

Now the syntax of FEAT-REP is shown. In this (formal) language a knowledge engineer can represent the experts knowledge about the structure hierachies and manufacturing qualities of workpieces. The characteristics of the features (feature interaction, different names, contextsensitivity, fragmentary description and abstract description) could be represented adequately.

What can be used as the quantitative level of FEAT-REP ? The *Boundary Representation* (*B-Rep*) serves as a basis for the most feature representations in the feature-based systems, i. e. the boundary surfaces of a workpiece are the atomic geometrical entities which are used to describe features. There are also efforts in research to use *Constructive Solid Geometry* (*CSG*) as the atomic geometrical entities (cf. [14]). In this paper TEC-REP serves as basis of FEAT-REP which is based on the B-Rep. FEAT-REP itself is a frame like language which is illustrated below.

Feature → Qualitative_Feature | Functional_Feature | Geometrical_Feature
Qualitative_Feature →
Featurename: featurename
Featuretype: featuretype
{Specialize_Feature: featurename}
{Subsumes_Features: (list_of_featurenames)}
{is_part_of: (list_of_featurenames)}
{has_parts: (list_of_featurenames) }
{Feature_Rule: (set_of_feature_graph_grammar_rules)}
{Rule_Attributes: (list_of_rule_attributes) }
{Embedding_Specifications: (list_of_embedding_specifications)}
{Described_Feature: featurename}
Description: (list_of_qualitative_constraints)
{Feature_Context: (list_of_context_constraints)}
Functional_Feature →
Featurename: featurename
Featuretype: featuretype
{Specialize_Feature: featurename}
{Subsumes_Features: (list_of_featurenames)}
{is_part_of: (list_of_featurenames)}
{has_parts: (list_of_featurenames) }
{Feature_Rule: (set_of_feature_graph_grammar_rules)}
{Rule_Attributes: (list_of_rule_attributes)}
{Embedding_Specifications: (list_of_embedding_specifications)}
{Described_Feature: featurename}
Description: (list_of_functional_constraints)
{Feature_Context: (list_of_context_constraints)}

Geometrical_Feature →
Featurename: featurename
Featuretype: featuretype
{Specialize_Feature: featurename}
{Subsumes_Features: (list_of_featurenames)}
{is_part_of: (list_of_featurenames)}
has_parts: (list_of_featurenames)
Feature_Rule: (set_of_feature_graph_grammar_rules)
Rule_Attributes: (list_of_rule_attributes)
{Embedding_Specifications: (list_of_embedding_specifications)}
{Feature_Context: (list_of_context_constraints)}

Featurename and Featuretype together identify the feature. The featurenames are given by the expert; the featuretypes are the differentations in the definition of the term feature, like *geometrical drilling* feature or *functional design* feature. Via Specialize_Feature and Subsumes_Features a hierachical structure over the features is constructed. This structure is generated and managed by a KL-ONE like conceptual language formalism called TAXON [2]. Is_part_of and Has_parts makes the part-of relation explicit. It is a redundant information like Subsumes_Features, too, and helps to make it easier to read the feature descriptions. Feature_rule is a set of alternative graph grammar rules which describes the featuregraph. Via this rule the parts of a feature are set into a (topological) relation. The attributes of a rule are divided via Rule_Attributes and Feature_Context into the attributes which depend only on the data of the featureparts itself and the attributes which depend on the data of the feature context. The context also includes informations about machines or tools. When Embedding_ Specifications is not specified, the default specification is used: Every adjacent edge to the right hand side of the rule is adjacent to the left hand side of the rule. Described_Feature is the link to the geometrical features but the underlaying geometry can also be described explicitly. Description is the list of the functional or qualitative constraints which describe the feature.

An example of the turning feature insertion is given below and illustrated in figure 4.

Featurename: insertion
Featuretype: geometrical turning feature
Subsumes_Features: (O-ring_groove)
is_part_of: (left_shoulder, right_shoulder, long_turning_surface, step)
has_parts: (left_shoulder, right_shoulder)
Feature_Rule: ((((insertion), (left_shoulder, right_shoulder)), ((insertion), (left_shoulder_insertion, right_shoulder))))
Rule_Attributes: (nil)

5. Feature Recognition

As proposed in the previous sections the importance of feature recognition in manufacturing stems from the fact that each feature can be associated with knowledge about how the feature should be manufactured; this information can be used to generate a process plan. From this point of view feature recognition forms a major component of the CAD/CAM interface for CAPP. In this paper we concentrated on the recognition of geometrical and qualitative features; the functional features are important for design only. Working with manufacturing features means to recognize these features from the CAD data to generate a working plan. Working with design features means to construct by means of these features and to expand them to the CAD data. The most recent developments in this research field can be read in e.g. [5] and [1].

Within our current research the features will be recognized or expanded by parsing methods which are based on graph matching methods and heuristics (background knowledge). This is facilitated through the representation of the feature definitions in a well-formed attributed node-label-controlled graph grammar. The *feature-parser* finds the complete set of features derivable from the productions of an ANLCGG given the grammar and a workpiece described in the terms of TEC-REP (an augmented topology graph representing the geometry and technology of a workpiece). So the problem of feature recognition is the problem of finding isomorphic subgraphs, in general a NP-complete problem. But it become solvable in $O(n^x)$ time using e.g. the method described in [10]: "*The technique is, to incorporate application dependent knowledge systematically ...*". The detail of the feature recognition algorithm will be published in a separate paper.

One problem that arises in the CAPP systems from the integration of CAD and CAM is that a workpiece must be transformed through different feature-languages, e.g. the one of a designer, a driller or a turner. On this way the workpiece passes different qualitative description languages. The gap between the single qualitative levels will be brigded by a quantitative description level, e.g. TEC-REP. This level contains all information needed to generate another qualitative description of the workpiece. But why forget the previous qualitative description? So when another qualitative description of the workpiece will be generated, the previous qualitative description could be used to make this generation more efficient. In figure 5 this method is illustrated.

For example when a designer constructs a Seeger circlip ring groove the same geometry can be seen as groove by the manufacturer; only the feature-names and the semantics must be changed. This method will be integrated in the feature recognition algorithm.

6. Conclusion

In our work we show that formal languages are useable to represent feature descriptions. The graph grammar based formalism FEAT-REP is a powerful and general tool to represent feature descriptions. A feature language defined in this fomalism represents a link between the quantitative (low-level) geometrical/technological representation and the qualitative (high-level) abstractions, as qualitative entities are expressed in terms of quantitative ones. Because the quantitative description of a workpiece can be seen as a topological graph, the features can be recognized by graph-based parsing.

In future research a domain dependent graph-based parsing algorithm will be developed. Currently, a small feature-grammar of one of our experts has been implemented using the extended D-PATR system, a formalism to represent unification-based grammars. Our quantitative representation formalism TEC-REP serves as a lexicon in this system.

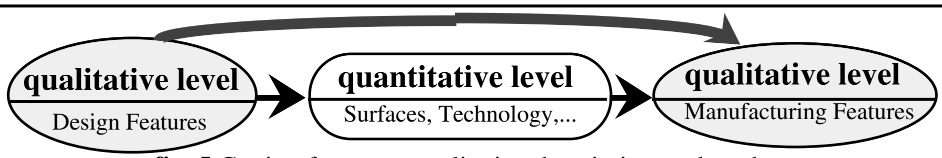


fig. 5 Getting from one qualitative description to the other

7. References

[1] Anderson, D.C. and Henderson, M.R.: *Computer Recognition and Extraction of Form Features: A CAD/CAM Link*. Computers in Industry (6) 4 (1984), 315-325.

[2] Baader, F. and Hanschke, P.: A Scheme for Integrating Concrete Domains into Concept Languages. In *International Joint Conference on AI,* 1991.

[3] Bernardi, A., Klauck, C., and Legleitner, R.: PIM: Planning in Manufacturing. In *forthcoming*, DFKIGmbH, 1991.

[4] Bernardi, A., Klauck, C., and Legleitner, R.: *TEC-REP: Repräsentation von Geometrie- und Technologieinformationen.* Dokument, D-91-07 Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, Postfach 20 80, D-6750 Kaiserslautern, june, 1991.

[5] Chang, T.C.: *Expert Process Planning for Manufacturing,* Addison-Wesley (1990).

[6] Cunningham, J.J. and Dixon, J.R.: Designing with features: the origin of features. In *International Computers in Engineering Conference and Exhibition,* july/august 1988, pp. 237-243.

[7] Dixon, J.R. and Finger, S.: *A Review of Research in Mechanical Engineering Design. Part II :Representations, Analysis, and Design for the Life Cycle*. Engineering DesignSpringer-Verlag New York Inc. (1) 2 (1989), 121-137.

[8] Friedland, P.E. and Iwasaki, Y.: *The Concept and Implementation of Skeletal Plans*. Journal of Automated Reasoning (1)(october 1985), 161-208.

[9] Goos, G. and Hartmans, J. (Eds). *Graph-Grammars and Their Application to Computer Science*. H. Ehrig and M. Nagl and G. Rozenberg and A. Rosenfeld, december 1986.

[10] Kaul, M.: Practical Applications of Precedence Graph Grammars. In *Graph-Grammars and Their Application to Computer Science,* Goos, G. and Hartmans, J., H. Ehrig and M. Nagl and G. Rozenberg and A. Rosenfeld, december 1986, pp. 326-342.

[11] Mullins, S. and Rinderle, J.R.: *Grammatical Approaches to Engineering Design, Part I: An Introduction and Commentary*. Research in Engineering Design 2 (1991), 121-135.

[12] Rinderle, J.R.: *Grammatical Approaches to Engineering Design, Part II: Melding Configuration and Parametric Design Using Attribute Grammars*. Research in Engineering Design 2 (1991), 137-146.

[13] Tengvald, E.: *The Design of Expert Planning Systems: An Experimental Operations Planning System for Turning*, Ph.D. dissertation, Lingköping University, Department of Computer and Information ScienceLinköping UniversityS-58183 Linköping, Sweden, 1984.

[14] Woodwark, J.R.: *Some speculations on feature recognition*. Computer Aided Design (20) 4 (may 1988), 189-196.

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

DFKI
-Bibliothek-
PF 2080
67608 Kaiserslautern
FRG

DFKI Publikationen

Die folgenden DFKI Veröffentlichungen sowie die aktuelle Liste von allen bisher erschienenen Publikationen können von der oben angegebenen Adresse oder per anonymem ftp von ftp.dfki.uni-kl.de (131.246.241.100) unter pub/Publications bezogen werden.
Die Berichte werden, wenn nicht anders gekenn-zeichnet, kostenlos abgegeben.

DFKI Publications

The following DFKI publications or the list of all published papers so far are obtainable from the above address or per anonymous ftp from ftp.dfki.uni-kl.de (131.246.241.100) under pub/Publications.
The reports are distributed free of charge except if otherwise indicated.

DFKI Research Reports

RR-92-49
Christoph Klauck, Ralf Legleitner, Ansgar Bernardi:
Heuristic Classification for Automated CAPP
15 pages

RR-92-50
Stephan Busemann:
Generierung natürlicher Sprache
61 Seiten

RR-92-51
Hans-Jürgen Bürckert, Werner Nutt:
On Abduction and Answer Generation through Constrained Resolution
20 pages

RR-92-52
Mathias Bauer, Susanne Biundo, Dietmar Dengler, Jana Koehler, Gabriele Paul: PHI - A Logic-Based Tool for Intelligent Help Systems
14 pages

RR-92-53
Werner Stephan, Susanne Biundo:
A New Logical Framework for Deductive Planning
15 pages

RR-92-54
Harold Boley: A Direkt Semantic Characterization of RELFUN
30 pages

RR-92-55
John Nerbonne, Joachim Laubsch, Abdel Kader Diagne, Stephan Oepen: Natural Language Semantics and Compiler Technology
17 pages

RR-92-56
Armin Laux: Integrating a Modal Logic of Knowledge into Terminological Logics
34 pages

RR-92-58
Franz Baader, Bernhard Hollunder:
How to Prefer More Specific Defaults in Terminological Default Logic
31 pages

RR-92-59
Karl Schlechta and David Makinson: On Principles and Problems of Defeasible Inheritance
13 pages

RR-92-60
Karl Schlechta: Defaults, Preorder Semantics and Circumscription
19 pages

RR-93-02
Wolfgang Wahlster, Elisabeth André, Wolfgang Finkler, Hans-Jürgen Profitlich, Thomas Rist:
Plan-based Integration of Natural Language and Graphics Generation
50 pages

RR-93-03
Franz Baader, Berhard Hollunder, Bernhard Nebel, Hans-Jürgen Profitlich, Enrico Franconi:
An Empirical Analysis of Optimization Techniques for Terminological Representation Systems
28 pages

RR-93-04
Christoph Klauck, Johannes Schwagereit:
GGD: Graph Grammar Developer for features in CAD/CAM
13 pages

RR-93-05
Franz Baader, Klaus Schulz: Combination Tech-niques and Decision Problems for Disunification
29 pages

RR-93-06
Hans-Jürgen Bürckert, Bernhard Hollunder, Armin Laux: On Skolemization in Constrained Logics
40 pages

RR-93-07
Hans-Jürgen Bürckert, Bernhard Hollunder, Armin Laux: Concept Logics with Function Symbols
36 pages

RR-93-08
Harold Boley, Philipp Hanschke, Knut Hinkelmann, Manfred Meyer: COLAB: A Hybrid Knowledge Representation and Compilation Laboratory
64 pages

RR-93-09
Philipp Hanschke, Jörg Würtz:
Satisfiability of the Smallest Binary Program
8 Seiten

RR-93-10
Martin Buchheit, Francesco M. Donini, Andrea Schaerf: Decidable Reasoning in Terminological Knowledge Representation Systems
35 pages

RR-93-11
Bernhard Nebel, Hans-Juergen Buerckert:
Reasoning about Temporal Relations:
A Maximal Tractable Subclass of Allen's Interval Algebra
28 pages

RR-93-12
Pierre Sablayrolles: A Two-Level Semantics for French Expressions of Motion
51 pages

RR-93-13
Franz Baader, Karl Schlechta:
A Semantics for Open Normal Defaults via a Modified Preferential Approach
25 pages

RR-93-14
Joachim Niehren, Andreas Podelski,Ralf Treinen: Equational and Membership Constraints for Infinite Trees
33 pages

RR-93-15
Frank Berger, Thomas Fehrle, Kristof Klöckner, Volker Schölles, Markus A. Thies, Wolfgang Wahlster: PLUS - Plan-based User Support
Final Project Report
33 pages

RR-93-16
Gert Smolka, Martin Henz, Jörg Würtz:
Object-Oriented Concurrent Constraint Programming in Oz
17 pages

RR-93-17
Rolf Backofen:
Regular Path Expressions in Feature Logic
37 pages

RR-93-18
Klaus Schild: Terminological Cycles and the Propositional μ-Calculus
32 pages

RR-93-20
Franz Baader, Bernhard Hollunder:
Embedding Defaults into Terminological Knowledge Representation Formalisms
34 pages

RR-93-22
Manfred Meyer, Jörg Müller:
Weak Looking-Ahead and its Application in Computer-Aided Process Planning
17 pages

RR-93-23
Andreas Dengel, Ottmar Lutzy:
Comparative Study of Connectionist Simulators
20 pages

RR-93-24
Rainer Hoch, Andreas Dengel:
Document Highlighting —
Message Classification in Printed Business Letters
17 pages

RR-93-25
Klaus Fischer, Norbert Kuhn: A DAI Approach to Modeling the Transportation Domain
93 pages

RR-93-26
Jörg P. Müller, Markus Pischel: The Agent Architecture InteRRaP: Concept and Application
99 pages

RR-93-27
Hans-Ulrich Krieger:
Derivation Without Lexical Rules
33 pages

RR-93-28
Hans-Ulrich Krieger, John Nerbonne, Hannes Pirker: Feature-Based Allomorphy
8 pages

RR-93-29
Armin Laux: Representing Belief in Multi-Agent Worlds viaTerminological Logics
35 pages

RR-93-33
Bernhard Nebel, Jana Koehler:
Plan Reuse versus Plan Generation: A Theoretical and Empirical Analysis
33 pages

RR-93-34
Wolfgang Wahlster:
Verbmobil Translation of Face-To-Face Dialogs
10 pages

RR-93-35
Harold Boley, François Bry, Ulrich Geske (Eds.): Neuere Entwicklungen der deklarativen KI-Programmierung — *Proceedings*
150 Seiten
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

RR-93-36
Michael M. Richter, Bernd Bachmann, Ansgar Bernardi, Christoph Klauck, Ralf Legleitner, Gabriele Schmidt: Von IDA bis IMCOD: Expertensysteme im CIM-Umfeld
13 Seiten

RR-93-38
Stephan Baumann: Document Recognition of Printed Scores and Transformation into MIDI
24 pages

RR-93-40
Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Werner Nutt, Andrea Schaerf: Queries, Rules and Definitions as Epistemic Statements in Concept Languages
23 pages

RR-93-41
Winfried H. Graf: LAYLAB: A Constraint-Based Layout Manager for Multimedia Presentations
9 pages

RR-93-42
Hubert Comon, Ralf Treinen:
The First-Order Theory of Lexicographic Path Orderings is Undecidable
9 pages

RR-93-44
Martin Buchheit, Manfred A. Jeusfeld, Werner Nutt, Martin Staudt: Subsumption between Queries to Object-Oriented Databases
36 pages

RR-93-45
Rainer Hoch: On Virtual Partitioning of Large Dictionaries for Contextual Post-Processing to Improve Character Recognition
21 pages

RR-93-46
Philipp Hanschke: A Declarative Integration of Terminological, Constraint-based, Data-driven, and Goal-directed Reasoning
81 pages

DFKI Technical Memos

TM-92-01
Lijuan Zhang: Entwurf und Implementierung eines Compilers zur Transformation von Werkstückrepräsentationen
34 Seiten

TM-92-02
Achim Schupeta: Organizing Communication and Introspection in a Multi-Agent Blocksworld
32 pages

TM-92-03
Mona Singh:
A Cognitiv Analysis of Event Structure
21 pages

TM-92-04
Jürgen Müller, Jörg Müller, Markus Pischel, Ralf Scheidhauer:
On the Representation of Temporal Knowledge
61 pages

TM-92-05
Franz Schmalhofer, Christoph Globig, Jörg Thoben:
The refitting of plans by a human expert
10 pages

TM-92-06
Otto Kühn, Franz Schmalhofer: Hierarchical skeletal plan refinement: Task- and inference structures
14 pages

TM-92-08
Anne Kilger: Realization of Tree Adjoining Grammars with Unification
27 pages

TM-93-01
Otto Kühn, Andreas Birk: Reconstructive Integrated Explanation of Lathe Production Plans
20 pages

TM-93-02
Pierre Sablayrolles, Achim Schupeta:
Conlfict Resolving Negotiation for COoperative Schedule Management
21 pages

TM-93-03
Harold Boley, Ulrich Buhrmann, Christof Kremer:
Konzeption einer deklarativen Wissensbasis über recyclingrelevante Materialien
11 pages

TM-93-04
Hans-Günther Hein: Propagation Techniques in WAM-based Architectures — The FIDO-III Approach
105 pages

DFKI Documents

D-92-24
Jürgen Müller, Donald Steiner (Hrsg.):
Kooperierende Agenten
78 Seiten

D-92-25
Martin Buchheit: Klassische Kommunikations- und Koordinationsmodelle
31 Seiten

D-92-26
Enno Tolzmann:
Realisierung eines Werkzeugauswahlmoduls mit Hilfe des Constraint-Systems CONTAX
28 Seiten

D-92-27
Martin Harm, Knut Hinkelmann, Thomas Labisch: Integrating Top-down and Bottom-up Reasoning in COLAB
40 pages

D-92-28
Klaus-Peter Gores, Rainer Bleisinger: Ein Modell zur Repräsentation von Nachrichtentypen
56 Seiten

D-93-01
Philipp Hanschke, Thom Frühwirth:
Terminological Reasoning with Constraint Handling Rules
12 pages

D-93-02
Gabriele Schmidt, Frank Peters, Gernod Laufkötter: User Manual of COKAM+
23 pages

D-93-03
Stephan Busemann, Karin Harbusch(Eds.):
DFKI Workshop on Natural Language Systems: Reusability and Modularity - Proceedings
74 pages

D-93-04
DFKI Wissenschaftlich-Technischer Jahresbericht 1992
194 Seiten

D-93-05
Elisabeth André, Winfried Graf, Jochen Heinsohn, Bernhard Nebel, Hans-Jürgen Profitlich, Thomas Rist, Wolfgang Wahlster:
PPP: Personalized Plan-Based Presenter
70 pages

D-93-06
Jürgen Müller (Hrsg.):
Beiträge zum Gründungsworkshop der Fachgruppe Verteilte Künstliche Intelligenz Saarbrücken 29.-30. April 1993
235 Seiten
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-93-07
Klaus-Peter Gores, Rainer Bleisinger:
Ein erwartungsgesteuerter Koordinator zur partiellen Textanalyse
53 Seiten

D-93-08
Thomas Kieninger, Rainer Hoch: Ein Generator mit Anfragesystem für strukturierte Wörterbücher zur Unterstützung von Texterkennung und Textanalyse
125 Seiten

D-93-09
Hans-Ulrich Krieger, Ulrich Schäfer:
TDL ExtraLight User's Guide
35 pages

D-93-10
Elizabeth Hinkelman, Markus Vonerden,Christoph Jung: Natural Language Software Registry
(Second Edition)
174 pages

D-93-11
Knut Hinkelmann, Armin Laux (Eds.):
DFKI Workshop on Knowledge Representation Techniques — Proceedings
88 pages

D-93-12
Harold Boley, Klaus Elsbernd, Michael Herfert, Michael Sintek, Werner Stein:
RELFUN Guide: Programming with Relations and Functions Made Easy
86 pages

D-93-14
Manfred Meyer (Ed.): Constraint Processing – Proceedings of the International Workshop at CSAM'93, July 20-21, 1993
264 pages
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-93-15
Robert Laux: Untersuchung maschineller Lernverfahren und heuristischer Methoden im Hinblick auf deren Kombination zur Unterstützung eines Chart-Parsers
86 Seiten

D-93-20
Bernhard Herbig:
Eine homogene Implementierungsebene für einen hybriden Wissensrepräsentationsformalismus
97 Seiten

D-93-21
Dennis Drollinger:
Intelligentes Backtracking in Inferenzsystemen am Beispiel Terminologischer Logiken
53 Seiten