GGD: Graph Grammar Developer for features in CAD/CAM

Christoph Klauck and Johannes Schwagereit

German Research Center for Artificial Intelligence Inc. (DFKI)
ARC-TEC Project
Mailing address: P.O. Box 2080, D-6750 Kaiserslautern
Telephone: ++49-631-205-3477
E-mail: klauck@dfki.uni-kl.de

Abstract

To integrate CA*-systems with other applications in the world of CIM, one principal approach currently under development is based on feature representation. It enables any CIM component to recognize the higher-level entities – the so-called *features* – out of a lower-data exchange format, which might be the internal representation of a CAD system as well as some standard data exchange format. In this paper we present a 'made-to-measure' editor for representing features in the higher-level domain specific representation language FEAT-REP – a representation language based on a (feature-) specific attributed node labeled graph grammar. This intelligent tool, shortly called GGD, supports the knowledge engineer during the representation process by structuring the knowledge base using a conceptual language and by verifying several characteristics of the features.

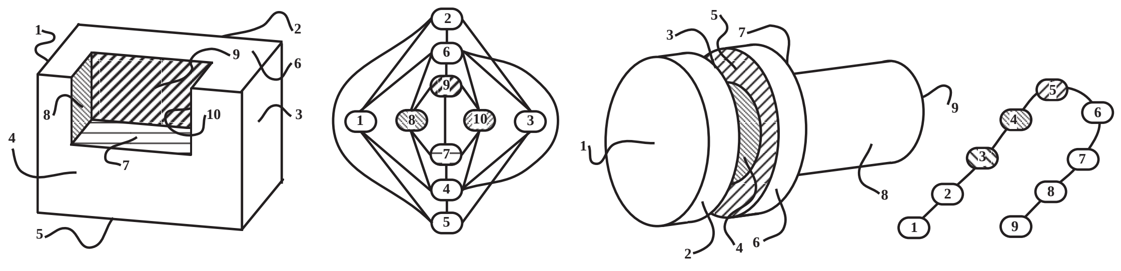


Figure 1: Neighborhoodgraphs of surfaces

Contents

List of Figures

1 Motivation

Research in feature-based CA*-systems like Computer Aided Design (CAD), or Computer Aided Process Planning (CAPP), has been motivated by the understanding that geometric models represent a workpiece in greater detail than it can be utilized e.g. by a designer or process planner. When CA*-experts look at a workpiece, they perceive it in terms of their own expertise – the so-called *features*. Features are domain- and company-specific description elements based on the geometrical and technological data of a workpiece that an expert in a domain associates with certain informations [2]. They are build upon a *syntax* (shape description: geometry and technology, given here by productions of a graph grammar) and a *semantics* (description of related informations, e.g. skeletal plans in manufacturing or functional relations in design) and they provide an abstraction mechanism to facilitate e.g. the creation, manufacturing or analysis of workpieces or more general to bridge the gap between several systems in the world of Computer Integrated Manufacturing (CIM). Features that are required e.g. for design may differ considerably from those required e.g. for manufacturing or assembly, even though they may be based on the same geometric and technological entities [6].

So representing features is one necessary step to bridge the gap between several CA*-systems and an important step towards truly Computer Integrated Manufacturing. The expected advantages of a close coupling of CA*-systems are: The information interchange shall lead to a better knowledge transfer, to shorter turnaround times and to improved feedback. At the end, higher Ωexibility and generally better results are expected.

In current research one method to represent features is based on graph grammars (cf. [3, 6, 15]). This area 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 [5]. So in consideration of the feature characteristics 'made-to-measure' tools must be developed to make the recognition and representation process more efficient.

From this point of view we present in this paper an implementation of the high level domain-specific feature representation language FEAT-REP [10]. This implementation is realized by the Graph Grammar Developer (GGD) – an intelligent tool to support users of FEAT-REP to fill the knowledge base with definitions of features.

2 What are Features ?

To become more familiar with the effect of feature characteristics to our representation formalism, we would like to introduce brieΩy the most important characteristics of its descriptions. Detailed explanations and the analogue to graph grammars can be found in [10]. Some of the most important syntactical characteristics of features are:

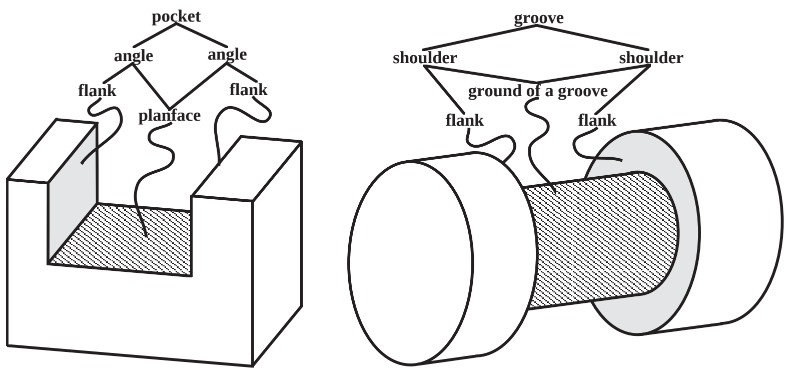


Figure 2: Overlapping of shoulders or angles

Similar definitions: In an application the knowledge base containing the feature descriptions will be large. Many of these descriptions will be similar to each other, descriptions for a single feature as well as those for various features, with respect to a *has-parts* (e.g. figure 3) and a *is-a* hierarchy.

Component overlapping: Features may have relations to features of different workpieces (e.g. bearing).

Dependence of Dimensions: In dependence of dimensions, the same structures may be identified as different features (e.g. *groove* and *insertion*).

Fragmentizing: Parts of features are not always in direct neighborhood (e.g. *FRAGM-LONGTURN* in figure 3).

Ambiguity: In the terminology of features an expert often have different alternative descriptions for the same structure (e.g. *groove* or *pocket*).

Neighborhood: Feature descriptions form graphs of features and/or surfaces (see figure 1), where edges represents the neighborhood.

Interaction: Areas of features can overlap (see figure 2).

Additional characteristics are *contextsensitivity* (e.g. *LONGTURN-OUT* and *GROUND-OF-GROOVE* in figure 3) and *defectivity*.

3 Attributed Node Labeled Feature Graph Grammars

In this section we will brieΩy define the terminology of attributed node labeled graph grammars as used in this paper. Introduction and survey can be found in more detail e.g. in [5].

In our paper the term *(feature-) graph* means an attributed finite undirected node labeled graph, in the sequel shortly called *graph*. Such a (feature-) graph FG is defined as a 4-tupel $FG := (V, E, \Sigma, \varphi)$, where V is a finite (nonempty) set of *attributed nodes*, $E \subseteq V \times V$ is a set of undirected *edges*, Σ is a finite (nonempty) alphabet of *node labels* or *sorts* and φ is a *labeling function*, with $\varphi : V \to \Sigma$. Workpieces are represented by such graphs. The nodes of a workpiecegraph represent geometric primitive surfaces, the node label decode the type of the surface (e.g. *cylinder jacket*), the attributes carry detailed geometric and technologic information (e.g. tolerances) and the edges decode the topology of the workpiece, i.e. two nodes are adjacent if the corresponding surfaces touch each other.

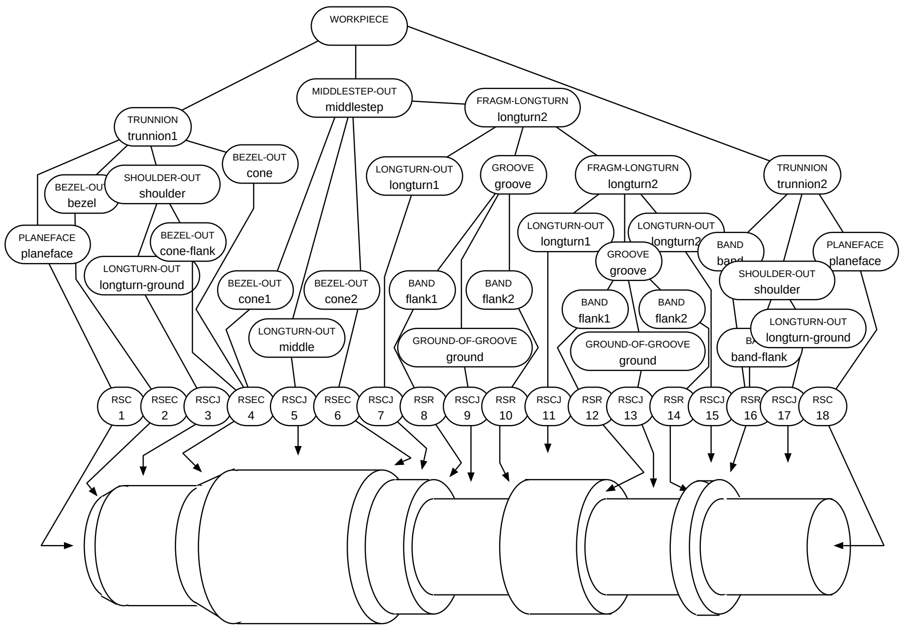


Figure 3: A workpiece and its feature structure

An *attributed node label (feature-) graph grammar* (ANLFGG) is a 4-tuple $GG := (T, N, P, goal)$, where T is a finite (nonempty) set of *terminals*, N is a finite (nonempty) set of *non-terminals*, P is a finite set of *productions* and $goal \in N$ is the *start node*. A *production* (rule) $p \in P$ is a 4-tuple $(lhs, rhs, \varepsilon, c)$ where $lhs \in N$ is a single node, the *left hand side* of p, rhs is a (nonempty) (feature-) graph over $T \cup N$, the *right hand side* of p, ε is an *embedding specification* and c is a finite set of *con-*

ditions over *lhs* and *rhs*, the so-called *dependency relations*. The conditions or the so-called *constraints* c serve two purposes: First to proof or generate informations by calculating attributes and second to lay down certain restrictions and attributes given by a description of a feature.

The most graph grammar formalisms are distinguished by the embedding specification ε. In our case we define ε in that way that always an edge in a (feature-) graph of a derivation step represents the neighborhood of the two incident nodes. For details of our ANLFGG and the analogue to features see [11] and [10].

4 System Architecture of GGD

In contrast to other more general tools editing graph grammars (cf. [7, 9]) the GGD is specialized to edit FEAT-REP – the 'made-to-measure' (feature-) graph grammar formalism. Figure 4 shows the most important components of GGD and their interrelations.

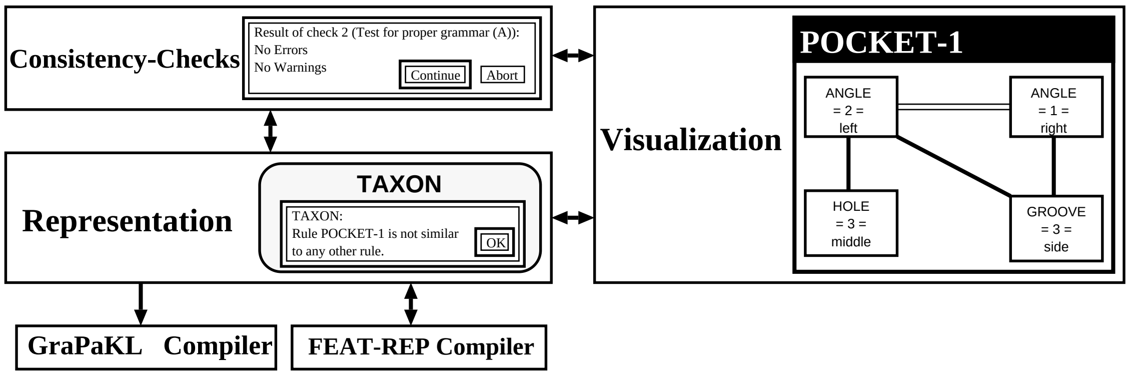
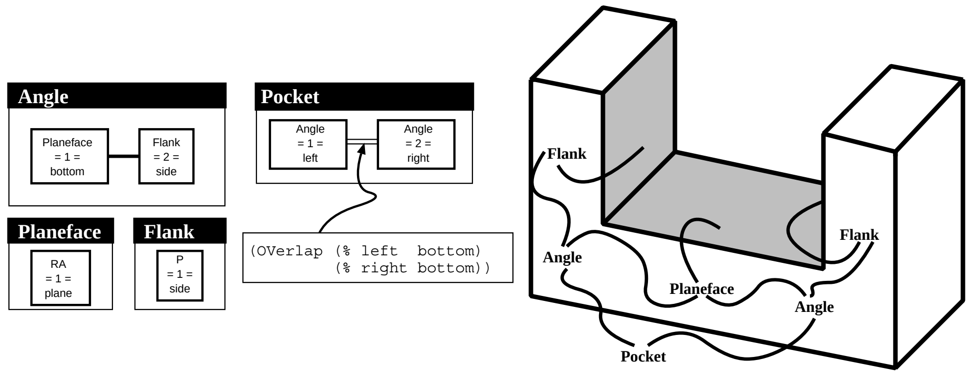


Figure 4: Structure of the GGD

The **visualization** component is the graphical user interface of the GGD. It offers the user an easy possibility to enter, view and manipulate definitions of features (see figure 8). A part of this component is a text-editor (*Constraints for ...*) to enter conditions. Using the designated menus all functions of the other components could be called. The GGD could also be used without taking advantage of the visualization component.

In figure 4 and 5 the visualization of a typical feature is shown. The user may add or delete nodes, neighborhoods and overlaps. For any of the nodes the sort has to be given, a label representing a second more specific name given by the user is optional but useful; the numbers are used for the parser GraPaKL as a kind of heuristics to specify an order in which he will try to find instances for the nodes. So they may change during the lifetime of the specified production. The optional labels support the descriptions of the conditions in a more natural way to identify the nodes in mind. Additional functions are provided to close or resize windows, or to move nodes and edges.

As shown the features are entered as graphs, which is a very abstract way to represent features. To give a more vivid illustration we currently develop a tool to show features as they would look as part of a workpiece. A first prototype is shown in figure 8.



```
(OVerlap (% left  bottom)
         (% right bottom))
```

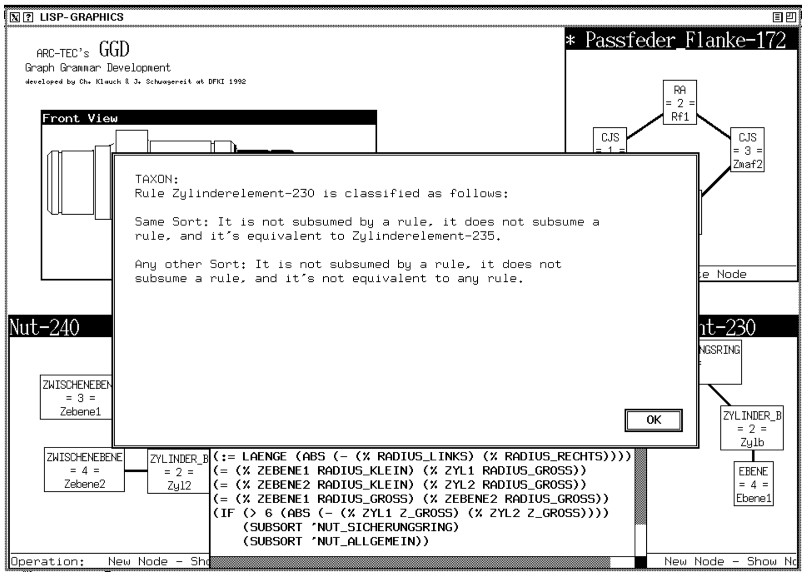
Figure 5: The rules forming a pocket

Figure 5 shows the feature *Pocket* and the corresponding features as they are represented by the GGD. Not shown in this illustration are any conditions belonging to the rules.

The **representation** component stores the entire knowledge. Several functions are provided for access and modification of the (feature-) graph grammar. Integrated in this component is a concept language based on KL-ONE (Knowledge Language ONE), called TAXON [8], which is developed for technical domains.

One drawback which concept languages based on KL-ONE have is that all the terminological knowledge has to be defined on an abstract logical level. In many applications like ours, one would like to be able to refer to concrete domains and predicates on these domains when defining concepts. Examples for such concrete domains are the integers, the real numbers or also non-arithmetic domains, and predicates could be equality, inequality or more complex predicates. TAXON realize a scheme for integrating such concrete domains into concept languages rather than describing a particular extension by some specific concrete domain. The used algorithms such as subsumption, instantiation and consistency are not only sound but also complete. They generate subtasks which have to be solved by a special purpose reasoner of the concrete domain [1].

TAXON is used to handle the feature characteristic of many similar definitions by defining a hierarchy of the productions. The right hand side of any production is compiled to a convenient form to be stored in TAXON. It was necessary to find a



```
TAXON:
Rule Zylinderelement-230 is classified as follows:

Same Sort: It is not subsumed by a rule, it does not subsume a
rule, and it's equivalent to Zylinderelement-235.

Any other Sort: It is not subsumed by a rule, it does not
subsume a rule, and it's not equivalent to any rule.
```

```
(:= LAENGE (ABS (- (% RADIUS_LINKS) (% RADIUS_RECHTS))))
(= (% ZEBENE1 RADIUS_KLEIN) (% ZYL1 RADIUS_GROSS))
(= (% ZEBENE2 RADIUS_KLEIN) (% ZYL2 RADIUS_GROSS))
(= (% ZEBENE1 RADIUS_GROSS) (% ZEBENE2 RADIUS_GROSS))
(IF (> 6 (ABS (- (% ZYL1 Z_GROSS) (% ZYL2 Z_GROSS))))
    (SUBSORT 'NUT_SICHERUNGSRING)
    (SUBSORT 'NUT_ALLGEMEIN))
```

Figure 6: TAXON in the GGD

representation which allows the concept language to compute exactly the subsumption hierarchy we respectively the expert have in mind. This has to be done efficient as the (feature-) graph grammar may be large. In TAXON a production a subsumes a production b, if b is an *expansion* of a, i.e. b can be generated out of a by inserting nodes into the right hand side of a.

According to the feature characteristic of many similar definitions TAXON hold to kinds of hierarchy: One for all feature definitions and one for every feature. Note that the latter is not just a part of the former.

Figure 7 shows a simple hierarchy of three features. *Shoulder-2* and *Shoulder-4* are expansions of *Shoulder-1*, *Shoulder-3* of *Shoulder-1* and *Shoulder-2*. It should be noted that our hierarchy is more extensive than just a subgraph relation. In future work the similarity of conditions will also be taken into account.

The GGD offers several **consistency checks** and verify the defined grammar for soundness. This will be performed during the development of a (maybe new) (feature-) graph grammar. The tests are adapted to our purpose, the aim is to prevent the description of features. This offers the user the possibility to detect and to eliminate the most errors as early as possible. Some of the performed tests are:

- A grammar can't be used without a start node. So it has to be checked if it has been defined and if it appear in any production on the right hand side. In the case of manufacturing or design features this should be a production for *workpiece*.
- Our definition requires that every feature graph is connected. Therefore the system checks if the productions right hand side is connected.

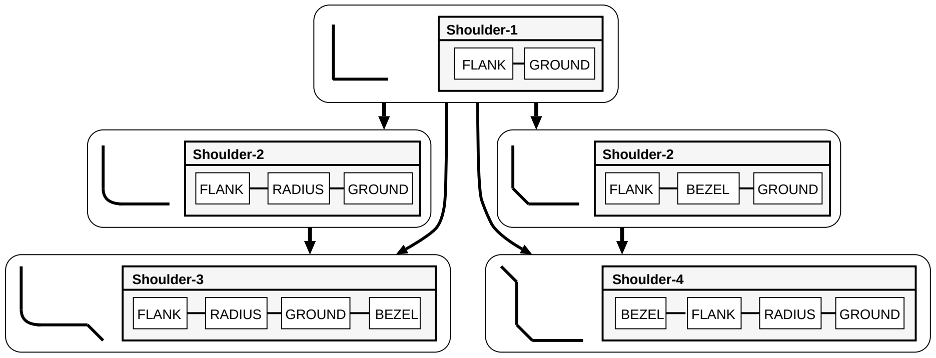


Figure 7: A hierarchy of features

- GGD verifies if the defined grammar sounds [4]. This verification contains the following checks:
  - There is no non-terminal (production) with an empty right hand side.
  - Every non-terminal can be expanded to a terminal graph. (Is there any senseless non-terminal ?)
  - For every non-terminal or terminal the start node can be expanded to a graph containing this symbol. (Is there any unreachable symbol ?)
- The GGD performs the task to check for a correct syntax of the conditions.
- A test is performed if every sort used by a production and its conditions is defined in the associated hierarchy. In addition GGD proof the hierarchy for cycle-free definitions.

Some checks are performed when a new or changed production and its associated conditions are saved by the user to the knowledge base. The complete check (see figure 3: *check rulebase*) is only performed on a request from the user.

The **FEAT-REP compiler** has the capability to read and to write files of this specific graph grammar formalism [10]. These files represent the knowledge base containing the descriptions of features. They are usable by programs for recognizing features (parse) and also by programs for feature based design (generate).

The program Graph Parser KaisersLautern (GraPaKL, [11]) is a heuristic driven chart based parser for our (feature-) graph grammars ANLFGG, adopted to recognize features of workpieces. The **GraPaKL compiler** translates the data stored in the representation component of GGD to files processable by the GraPaKL, say to its internal representation formalism. GraPaKL realize an abstraction step by transforming the geometrical and technological description of a workpiece into the qualitative level of the feature terminology. As result a feature structure is expected (see e.g. figure 3).

5 Developing a Feature Graph Grammar with GGD

The most important components of our graph grammar ANLFGG are the set of productions specifying the feature definitions and a hierarchy of sorts where every production is associated to one sort. If a production in the GGD is defined without specifying the associated sort, GGD automatically prompt an editor for defining it.

To develop a feature graph grammar the following sequence of steps is recommend to be performed:

Define the set of sorts specifying the super- and subsort relations. Querying the consistency check for the knowledge base maybe defined cycles will be found. Additionally GGD will point out, that there are no associated productions.

Define the set of productions. A copy-function can be used to specify similar rules. Also all conditions associated to a production have to be specified. After defining a production, GGD automatically check the (syntactical) correctness of this production. Also it is possible to check the classification of this production by TAXON.

Perform the consistency check for the grammar. After defining the set of sorts and the set of productions, during 4 stages the integrity of the knowledge base is checked. Errors or Warnings are maybe given by GGD.

Save the defined grammar in a FEAT-REP file. This file can be read again by GGD to modify the defined grammar or to generate a file for the parser. So a knowledge base have not to be defined in one session; interruptions are possible even though some errors occur during the previous step. GraPaKL files should be saved only if there are no errors in the knowledge base.

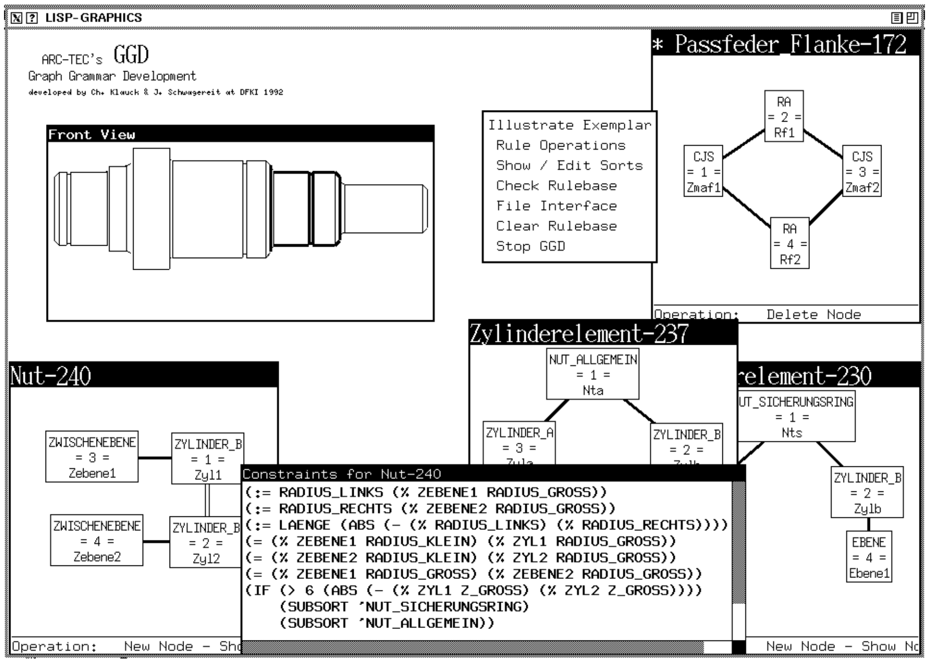
A successful feature graph grammar provided the drawing up (as a kind of a knowledge acquisition step) of a catalog containing the feature descriptions (syntax and semantics) in an informal manner. From one's own experience a typical sketch of the described features make this step more easy and more effective. It is important that this step is performed together with a knowledge engineer or at least by using domain specific acquisition tools (e.g. [13, 16]).

After describing the feature graph grammar GraPaKL is recommend to be used for checking the knowledge base of features on concrete workpieces. This test may show that there are still some errors in the descriptions of features which appear only during the runtime of GraPaKL and that some descriptions are incomplete, say that GraPaKL recognize not the intend features

6 Conclusion

We introduced an intelligent system to support the representation and the developing of features in CAD/CAM. 'Made-to-measure' graph grammars are used as a formal foundation, which is well suited, to represent the characteristics of features. Our tool GGD to edit our ANLFGG's should be efficient enough to handle even large and sophisticated (feature-) knowledge bases. The computation of hierarchies and the enforcement of several integrity checks make an efficient development of the grammar possible.

The knowledge representation and the integration of TAXON are already implemented. Until today this system is used by our CAPP-system called PIM (Planning In Manufacturing, [12]) to generate and maintain the knowledge base for manufacturing features. But it is also usable as domain independent editor for the specified graph grammar.



```
(:= RADIUS_LINKS (% ZEBENE1 RADIUS_GROSS))
(:= RADIUS_RECHTS (% ZEBENE2 RADIUS_GROSS))
(:= LAENGE (ABS (- (% RADIUS_LINKS) (% RADIUS_RECHTS))))
(= (% ZEBENE1 RADIUS_KLEIN) (% ZYL1 RADIUS_GROSS))
(= (% ZEBENE2 RADIUS_KLEIN) (% ZYL2 RADIUS_GROSS))
(= (% ZEBENE1 RADIUS_GROSS) (% ZEBENE2 RADIUS_GROSS))
(IF (> 6 (ABS (- (% ZYL1 Z_GROSS) (% ZYL2 Z_GROSS))))
    (SUBSORT 'NUT_SICHERUNGSRING)
    (SUBSORT 'NUT_ALLGEMEIN))
```

Figure 8: User interface of GGD

Future extension will be additional semantics checks, an improved user interface and a tool to generate graphs representing workpieces. GGD will also be integrated with the editor V-SKEP-EDIT [18] to offer the possibility of describing features and the associated skeletal plans in one session. Also a visualization of the defined features as shapes will be generated in future research. Figure 8 illustrate the today implemented user interface of GGD. In one window the user can highlight the features on the workpiece recognized by GraPaKL – the feature recognizer.

Currently GGD was used by a mechanical engineer to specify design features [17].

The training period takes about one week. No special knowledge about TAXON and the semantics checks was needed. Just the syntax of the language to specify the conditions of the features (*Constraints for* ...Window) which is like COMMON-Lisp takes a little bit time to learn.

7 Acknowledgements

This research was performed during the ARC-TEC project (Acquisition, Compilation and Representation of TEChnical knowledge) at the German Research Center for Artificial Intelligence Inc. (DFKI). It has been supported by grant from The Federal Ministry for Research and Technology (FKZ ITW-8902 C4).

References

[1] Baader, F. et al: *A Scheme for Integrating Concrete Domains into Concept Languages.* in: 12th IJCAI, pp. 452-457, 1991.

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

[3] Chuang, S.-H. et al: *Compound Feature Recognition by Web Grammar Parsing.*, in: Research in Engineering Design, Springer-Verlag, pp. 147-158, 1991.

[4] Drobot, V.: *Formal Languages and Automata Theory.*, Computer Science Press, Freeman & Co., 1989.

[5] Ehrig, H. et al: *Graph Grammars and Their Application to Computer Science.*, 1th - 4th International Workshop, Springer Verlag, LNCS 73, 153, 291, 532, 1979-1990.

[6] Finger, S. et al: *Parsing Features in Solid Geometric Models.*, in: ECAI'90, pp. 566-572, 1990.

[7] Goettler, H.: *Graphgrammatiken in der Softwaretechnik.*, Informatik Fachberichte 178, Springer-Verlag, 1989.

[8] Hanschke, P. et al: *TAXON: A Concept Language with Concrete Domains.* in: PDK'91, Springer-Verlag, LNAI 567, pp. 411-413, 1991.

[9] Himsolt, M.: *An Interactive Tool for Developing Graph Grammars.* in: 4th International Workshop of [5], pp. 61-65, 1990.

[10] Klauck, Ch. et al: *FEAT-REP: Representing Features in CAD/CAM.*, in: IV ISAI: Applications in Informatics, pp. 158-168, 1991.

[11] Klauck, Ch. et al: *A Heuristic Driven Parser Based on Graph Grammars for Feature Recognition in CIM*, in: SSPR'92, pp. 200-210, 1992.

[12] Klauck, Ch. et al: *Heuristic Classification for Automated CAPP.*, in: 11th AAI-XI'93, pp. *forthcoming*, SPIE, 1993.

[13] Kuehn, O. et al: *Integrated Knowledge Acquisition from text, previously solved cases and expert memories.*, in: Applied Artificial Intelligence, vol. 5, pp. 311-337, 1991.

[14] Mullins, S. et al: *Grammatical Approaches to Engineering Design, Part I: An Introduction and Commentary.*, in: Research in Engineering Design, Springer-Verlag, pp. 121-135, 1991.

[15] Rinderle, R.: *Grammatical Approaches to Engineering Design, Part II: Melding Configuration and Parametric Design Using Attribute Grammars.*, in: Research in Engineering Design, Springer-Verlag, pp. 137-146, 1991.

[16] Schmidt, G.: *Knowledge Acquisition form Text in a Complex Domain.*, in: 5th IEA/AIE-92, pp. 529-538, 1992.

[17] Schulte, M. et al: *Recognition of Design Features from Product Models.*, in: 9th ICED'93, pp. *forthcoming*, 1993.

[18] Wu, Z. et al: *Skeletal Plans Reuse: A Restricted Conceptual Graph Classification Approach.*, in: 7th AWCG, pp. 142-152, 1992.