Knowledge-based systems programming for knowledge intensive teaching

Citation for published version (APA):
KNOWLEDGE-BASED SYSTEMS PROGRAMMING
FOR KNOWLEDGE INTENSIVE TEACHING

Henri Achten, Jan Dijkstra,
Robert Oxman, Thijs Bax

Building Information Technology Notes
(BIT Notes)

This is a series of notes of the
section Bouwinformatica of the
faculty of Architecture, Building and Planning
Eindhoven University of Technology.
Since many of these are preliminary versions,
or may be published elsewhere, they have a
limited distribution only and are not for review.

ISSN 1381-3129

editors:
ing. Jan Dijkstra
ir. Jos P. van Leeuwen
prof. dr. Robert M. Oxman
prof. ir. Harry Wagter

Copies can be ordered from:
Secretary of the section Bouwinformatica,
Eindhoven University of Technology
Faculty of Architecture, Building and Planning
Postvak 20
P.O.Box 513
5600 MB Eindhoven
The Netherlands

© 1995
Eindhoven University of Technology,
faculty of Architecture, Building and Planning,
section Building Information Technologies.
Other issues in the series of BIT Notes are:


95/1 A Proposal for a distributed Virtual Reality system for Architectural simulations. A.H.A. Brouwer, W.A.H. Roelen, J.A.G. Jess

95/2 A Feature based approach to modelling Architectural Information. Jos P. van Leeuwen, Harry Wagter, Robert M. Oxman

Knowledge-based Systems Programming for Knowledge Intensive Teaching

Henri Achten, Jan Dijkstra, Robert Oxman, Thijs Bax

Keywords: Knowledge-based systems; Typological knowledge.

© 1995 Technische Universiteit Eindhoven, faculteit Bouwkunde, sectie Bouwinformatica. Alle rechten voorbehouden. Niets uit deze uitgave mag worden verveelvoudigd en/of openbaar gemaakt door middel van druk, fotocopie, microfilm, elektronische media of op welke andere wijze ook, zonder voorafgaande schriftelijke toestemming van de uitgever. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, transmitted or utilized in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior written permission of the publisher.
Abstract

This note gives an outline of a new pedagogical approach in a course teaching typological knowledge through the medium of Knowledge-Based Systems programming. It demonstrates how Knowledge-Based Systems offer an appropriate structure for analysing the knowledge required to implement typological design. Typological design implies extensive knowledge of building types in order to design a building belonging to a particular type. Such knowledge facilitates the design process, which can be considered as a sequence of decisions.

The building type used in the course is the office building. The implementation language is AutoLISP which operates in the AutoCAD-environment. In order for students to become acquainted with both building type and programming in AutoLISP, information and instructions have been gathered and prestructured, including a worked out example and AutoLISP code. Office plans are generated through use of the Knowledge-Based System. They are encoded in the form of frames. A specific technique for analysis of the building type and structuring the Knowledge-Based System is demonstrated.

The class consists of third-year undergraduate students with no extensive previous programming experience. At the end of the course the students will have learned the basics of Knowledge-Based Systems, have been introduced to programming these systems, have analysed and reflected upon the typological design process, and gained insight into a specific building type.
KNOWLEDGE-BASED SYSTEMS PROGRAMMING FOR KNOWLEDGE INTENSIVE TEACHING

Introduction

This note presents a new pedagogical approach to a computer-integrated design education course titled Knowledge-Based Systems in Building. The Introduction section will elaborate on the didactic issues of the course. In Knowledge-Based Systems, a brief discussion introduces some basic terminology. The next section, Typological Knowledge, will deal with the knowledge aspect of a specific building type. Analysing and Structuring Typological Knowledge discusses the specific approach towards the application of typological knowledge and Knowledge-Based Systems in the course. The particular implementation into the system is discussed in Implementing the Knowledge-Based System. A Students Example shows results of the work. The note ends with Conclusions, and Listings of Lisp-files.

The course Knowledge-Based Systems in Building consists of lectures and programming classes. The lectures cover basic principles of Knowledge-Based Systems. Subjects covered by the lectures are, among others, knowledge, knowledge acquisition, knowledge representation, knowledge bases, representing designs and design knowledge, and modelling reasoning in design. Some techniques used in expert systems and other AI technologies are also studied. The theoretical material of the lectures will be the subject of the exercise. In the exercise, students are required to develop a Knowledge-Based System that applies knowledge from the field of building. The programming will be done in AutoLISP in the AutoCAD environment. Most students do not have extensive programming experience. Therefore, the start of the exercise is devoted to an introduction to AutoLISP programming.

The course is part of a series of four courses which are compulsory for students who want to graduate in Building Information Technology (BIT). The other three courses of the series are: Advanced Architectural Modelling in CAD-systems; Advanced Architectural Representation in Multi-Media; and Information Systems in Building. Students following Knowledge-Based Systems in Building are required to have first followed Information Systems in Building. The course is given by a member of the BIT Group and a member of the GOM Group (Design Methods Group). Approaches in the course reflect ongoing research. Part of the theoretical lectures are given by guest-lecturers.

In addition to BIT students, the course is open to any student in architecture. The course is usually attended by third and fourth-year undergraduate students. It aims to introduce students to matters of knowledge acquisition, knowledge representation in information systems, and knowledge of design models. At the end, the students should have learned enough to be able to participate in projects that include knowledge acquisition for information systems development. The goal from the design point of view is to introduce the concepts of typological knowledge, structured analysis, and design reasoning, and their significance in the design process.
Knowledge-based systems

Knowledge-Based Systems attempt to capture and render operable human knowledge about some domain. Although they do not aim to mimic human cognitive processes, they provide structures that enable human beings to use them in ways that are intuitively appealing. The goal is to assist users in executing tasks that usually have a problem-solving character. Through making knowledge comprehensible both to machine and human, Knowledge-Based Systems can provide understanding in these processes. A motive for using Knowledge-Based Systems in design is the possibility for increasing productivity by automating low-level design decision making. Through implementing Knowledge-Based Systems, knowledge that often is obscure and ill-defined becomes more explicit. It provides insight into reasoning mechanisms that are or are not used. The major characteristic of Knowledge-Based Systems, in contrast with conventional programs, lies in the separation between knowledge and reasoning. That is, the system is equipped with a general-purpose reasoning facility (inference engine), which is capable of reasoning with pieces of knowledge it either has or acquires (for a general discussion of Knowledge-Based Systems, see Coyne et al. 1990).

Knowledge can be defined as anything someone knows about a certain subject, or as a reasoning model through which data are used in order to obtain new data or a new reasoning model. Knowledge is distinguished in several ways, one of which is between heuristic and deep knowledge. Heuristic knowledge concerns the ability to solve complex problems, usually based on experience. Deep knowledge is explicit, and is usually only acquired through study. Deep knowledge often is derived from sources. Another distinction often made is between procedural and declarative knowledge. Procedural knowledge concerns the 'how' of knowledge, declarative knowledge concerns the 'what' of knowledge (see Akin 1986).

A Knowledge-Based Design System stores and uses knowledge of a specific domain. Part of this knowledge refers to relevant concepts, facts, and objects. The list of concepts can be compared with the declaration of variables in a conventional program. This declarative knowledge in a design system is static. It can be expanded with dynamic information; procedural knowledge.

Typological knowledge

Implementing a Knowledge-Based System in an architectural context requires an architectural subject. In previous editions of the course, the case of kitchen-design was used. In order to provide a new emphasis in the course on knowledge formulation, the focus of the case was changed into typological design. Typological design requires extensive knowledge of building types. Architects draw from this knowledge in order to design a building belonging to a type. Even when the design task does not require a specific building type, architects utilise knowledge present in building types to inform their activities.

According to Heath (1984), the complexity of the design task—in relation to building types—can be differentiated into three classes of buildings: commodity, symbolic, and systems buildings. Commodity buildings constitute a class of building of which there exists a high degree of consensus on appearance, use, and design (for example office buildings, mass housing, etc.). Symbolic buildings usually are designed according to non-functional requirements as their representational function is
of primary consideration (for example churches, museums, etc.). Systems buildings are highly complex design tasks the result of which is always unique for the task at hand (for example hospitals, university campuses, etc.). Of these three, commodity buildings are well-constrained building types of which there is common agreement. This class of buildings is relatively easy to analyse and produce. Office buildings can be considered commodity buildings and were selected as the subject for implementation in the Knowledge-Based System.

The students have to become acquainted with the building type in a short space of time. Therefore, material concerning office buildings is gathered for them beforehand in a literature survey. Statements concerning major decisions in the design of office buildings are extracted. The statements are selected for their application to medium-height office buildings. They are presented as a series of statements from each source (Figure 2a; each statement is recorded in the order in which it is found in the source, and numbered accordingly), and ordered on the basis of subject category (Figure 2b; each source has a short code, after which the number of each statement in that source follows). In the first way of presenting the material, students are able to consult knowledge from each source separately. In the second, they are presented with the material ordered per subject (e.g., building, organisation, spaces, structural system, circulation, HVAC, etc.). Although the sources are taken from current work, they do range in age. Also they differ in their context (European and American sources). This diversity of sources is necessary because no single source covers all aspects of the building type. Different sources, therefore, add missing information. The material forms part of the knowledge base from which students implement the Knowledge-Based System. The particular approach in the course will next be dealt with.
NEUFERT BAUENTWURFSLEHRE

1. Raumgruppen: Bürobereich: Zellenbüros 1:3
   Personen mit AZUBI Arbeitsplätzen. Gruppenbüros bis
   20 Personen mit AZUBI Arbeitsplätzen. Grobgruppen
   bis 200 Personen auf einer Ebene. Kombibüros mit
   Einzelarbeitsplätzen und gemeinschaftlich nutzbaren
   Gruppenbereichen.

2. Durch Bildschirmarbeitsplätze (n, Gottschalk 1990
   ca. 20%, lit. BAP) und damit Computerterminals und
   Zusatzgeräte, steigt der Flächenbedarf für den
   Bürowerksplatz zunächst additiv um ca. 2-3 m² auf
   ca. 15-18 m² an.

3. 75% der täglichen Arbeit findet am "Engeren und
   Erweiterten Arbeitsplatz" statt. Notwendige
   Arbeitskontakte, sowie kollektiv genutzte Einrichtungen
   sind von Bedeutung. Daher die Forderung einer
   Nutzungsgestaltung aus Einzel- und Gruppenräumen,
   "Persönliche" und "Kollektive" Arbeitsplätze.

4. Nach Rosenauer bei 90% aller Bürogebäuden in USA
   Hauptachse OW, da tief einpringende Morgen- und
   Abendsonne statt. Südosten ist durch Sonnenschutz
   leicht abzuschirmen. Nach Joodieke Hauptachse SN
   Lage, damit Durchschnüren aller Räume gewährleistet
   ist. Räumle ist nur bei fünf Frei Anlagen vertretbar.

5. Einbündige Anlagen unvorteilhaft; nur bei liefern
   Büroarkumen vertretbar.

6. Zweibündige Anlagen bisher mehrheit der
   Verwaltungsgebäuden, Einzelräume und kleine Bürosäle

BUILDING

Neufert 6Orientation E/W in USA, S/N in Europe
Neufert 8 Daylight useful as far as 7 m depth in room
Neufert 10 Daylight possible through end facades, reclining
   facade and light-coves
Neufert 11 Daylight depth T = 1.5 Hf (height window)
Neufert 14 High-rise = building with highest floor higher than 22
   m above ground-level
Integration Office buildings typically have a Total Surface /
   Rental Surface ratio of 1:35
Offices 2 Four classes of depth in wings: shallow (4-5 m), semi
   shallow (6-10 m), deep (11-19 m), and very deep (20- m)
Update 1 Trend towards shallow small offices
Update 3A Half-shallow offices optimal depth (14.17 m)
Time 13 Area within 7.62-9.14 m of the facade best rentable,
   therefore offices usually are slabs, 18.29-21.34 m
Entwurf 2 Minimal feasible surface area 600 m²

ORGANISATION

Update 275% of the workers do not require flexibility
Time 3 Executive core concept

Analysing and Structuring Typological Knowledge

The building type provides the information required to develop the knowledge
base of the Knowledge-Based System. The declarative content of a type has been dealt
with above. Instantiation of a building from a type is generally conceived of as a
process of particularisation. In this view, an abstract type holds all generalised
knowledge necessary to generate a building by specifying its general aspects (Gero
1990; Coyne et al. 1990; Schmitt 1993). Although the relation between an abstracted
type and a typical instance makes logical sense, it is altogether not clear how a
complex and highly specific entity such as a building design can be derived from a
process of particularisation only (without making the type-object unmanageably
large). There is no direct mapping from the context-free abstract type to the site and
program specific building.

Considering types as declarative knowledge objects, therefore, does not seem
the right way to gain a better understanding of types in design. Furthermore, it goes
against the well established observation that as designers become more experienced,
they rely more on procedural knowledge than on declarative knowledge (Akin 1986).
It seems to make more sense to concentrate on procedural characteristics of the use of
types rather than declarative. In order to distinguish between these two approaches, it
is proposed to discuss "typological knowledge" rather than "types".
When the architectural design process is observed, it appears that the architect uses graphic representations (sketches and drawings) extensively to develop the design. Graphic representations denote the state of the design object, they inform other design participants, and they provide the architect with new input. Because of conventions of depiction, these representations can be interpreted by other members of the design community. The use of graphic representations as a way of encoding knowledge of the design object in the design process points to an important direction of development. At the same time, in the knowledge point of view, they complement the textual character of the declarative knowledge content discussed above. This is important since topological and formal matters are better discussed graphically rather than textual. These matters form also part of the declarative knowledge base of the designer.

The design process can be considered as a sequence of decisions about the design object. Decisions in the process are influenced by the program of demands, the site, design party preferences, regulations, and the way the design proceeds itself. The type as knowledge of a class of buildings, obviously informs the design process as well. Combined with the previous discussion on graphic representations, these notions result in considering design as a process that can be described through a sequence of well-defined graphic representations with specific knowledge-contents concerning the design object. Each representation can be distinguished on what it defines of the design object it represents. We refer to these specific representations as "generic representations".

The icons in Figure 3 signify in a graphic form the essence of each generic representation. They have a specific name which also describes the content of the generic representation. When using the “Platonic form”, a designer makes decisions about the general shape of the building, its main organisation and massing. In “Specified form”, tentative dimensions are given to the shape. Decisions about the internal organisation are taken in “Zone representation without contour”. Differentiating between specific areas in zones happens in “Schematically subdivided zone”. Specifying possible functions or uses of the areas occurs in “Function symbols in schematically subdivided zone”. Placing this information in the established form results in “Zone representation in specified form”. Formulated in this way, the sequence of decisions that are taken in the design process are teased from each other through specifying the accompanying graphic operations.

A sequence of generic representations, therefore, offers a procedural structure to analyse a specific building type. In order to demonstrate how such an analysis may take place, the students are provided with a worked out sequence of a specific subtype of the office building: the T-Shape. This sequence will be briefly discussed next. In each step of the sequence, the graphic representation encodes more aspects of the design object. This is done on the basis of the knowledge base provided. Together
with a simplified program of demands and a site, it is possible to both denote the pieces of knowledge required to make decisions, as well as to denote the reasoning mechanisms necessary to come to these decisions. In the example for the students, the use of knowledge and reasoning mechanisms are elaborated. The example also provides them with the implementation approach taken in the exercise towards Knowledge-Based Systems.

**PLATONIC FORM**

Subtype: T-Shape

Program of demands requires 7500 m\(^2\) area. The site is 75 \times 75 m^2.

Area S:
- 8 \times 940 = 7520 m^2
- 7 \times 1075 = 7525 m^2
- 6 \times 1250 = 7500 m^2
- 4 \times 2000 = 8000 m^2
- 3 \times 2500 = 7500 m^2
- 2 \times 3750 = 7500 m^2
- 1 \times 7500 m^2

Storey height: 3.10 m

**SPECIFIED FORM**

Determine orientation:

First estimate of dimensions: if A=B=C=D=E then A = \text{square root} \ S/4

S = 1500 m^2, therefore A=B=C=D=E=19.36 m

Choose depth of the wing: 22-38 m

Determine margins of wing length:

If A=B=C and B=D=12 m; A=37.7 m

If A=B=C and B=D=20 m; A = 8.3 m

Wing depth: [18.30 ; 37.70] or Length: [56.70 ; 85.31]

Wing length = 30 m, Wing depth = 15 m

Fig. 4 (left) - Analysis of T-Shape office building in “Platonic form”.

Fig. 5 (right) - Analysis of T-Shape office building in “Specified form”.

Figures 4 and 5 show the analysis taken in the first two generic representations. They demonstrate how decisions are taken on the basis of making drawings and consulting the knowledge base. Moving from the first step in “Platonic form” (the empty T-shape) to the second step (T-shape with “S”) requires knowledge of prevailing floor-areas in office buildings. At this point it is also possible to choose a tentative height of the storey. In the next step, the form is given parameters which in fact defines what aspects of the form can be changed. This completes the first generic representation. In “Specified form”, orientation is determined first. In order to decide on the dimensions of the shape, a first insight into the magnitude of the dimensions is made on the basis of the assumption that all parameters are equal. This information helps to decide on the class of wing depth (based on data in the knowledge base). Choosing a class results in a range of dimensions for the length and the depth of the wings. “Specified form” ends when the dimensions are tentatively appointed.
The choice of a zoning structure determines the internal organisation of the building, and is also dependent on the choice of wing depth. Several kinds of zoning can be distinguished, and their dimensions can be estimated on the basis of requirements of office depth and circulation area. This is what is established in “Zone representation without contour” (Figure 6). “Schematically subdivided zone” is concerned with identifying areas with specific properties within the zoning. In the case of the T-Shape office: end of wing, internal corners, and crossing zones. These areas subdivide the zone. They are indicated schematically in the generic representation (Figure 7).

The final step (see Figure 8; next page) for the exercise concerns assignment of functions to the zoning in the form. This will result in a schematic plan of which the dimensions, orientation, organisation, and functions have been determined. Throughout this process of analysis, the required pieces of knowledge from the knowledge base have been identified, and the required reasoning mechanisms have been defined. Since every step has been visualised explicitly through drawings, it is a process that is also readily accessible for a designer. It provides both the structure for analysis of a building type as for building the Knowledge-Based System.

It is important to note that although the approach is based on the nature of architectural design, it does not aim to model the actual cognitive activity of the architect. Obviously, the required explicitness demands a definition of each single decision, whereas an architect would handle a mixture of a multitude of generic representations and decisions in one drawing. Rather, the approach constitutes a way to establish a structure in which typological knowledge can be applied in ways that can be made intelligible for both computer and man.
Function symbols in schematically subdivided zone

![Function symbols in schematically subdivided zone](image)

G - Group office (-20 p)
T - Open plan office (20- p)
C - Circulation
R - Reception
k - Office cells

Fig. 8 - T-Shape office building in “Function symbols in schematically subdivided zone”

Implementing the Knowledge-Based System

The specific form chosen for representing knowledge in a Knowledge-Based System is the frame. A frame is a structured description that consists of a number of slots that can be filled in. The slots represent properties of the specific object being represented. The content of a slot can be a value, or an index or reference or an instruction to find a value for that specific aspect of the object. The instruction to find a value typically is in the form of a program which runs when a specific situation occurs (see Coyne et al. 1990 for a general discussion of frames). An empty defined frame (that is, a frame with slots that are not filled in) is usually referred to as a class frame, and a completely filled in frame is usually referred to as an instance frame. The actual design made in the system can be stored as a drawing file, but also as an instance frame. In this way, a case base of office designs can be made which is more compact than a similar set of drawing files. Regenerating a design from the case base follows from interpreting the slots of the instance frame. The frame technique is a flexible approach to represent knowledge because the content of a frame is not fixed and depends on the use it is put to.

The example of Figure 9 (next page) shows a frame named ‘office’, which has the following slots: ‘Name’, ‘Type’, ‘Total area’, ‘#Stories’, ‘Storey height’, and ‘Data’. In this example, the slots ‘Name’, ‘Type’, ‘#Stories’, and ‘Storey height’ have been filled in with values. The slot ‘Total area’, in order to be filled in, calls the program code “def_area” to obtain a value. The slot ‘Data’, which contains the specific information of the T-Shape type, contains another frame called “Tshape”.

11
The frame has been employed as the basis of the Knowledge-Based System in the exercise. It represents the general description of the office building type. The slots and the order in which they are evaluated constitute the procedural knowledge of the office building type. The values of the slots constitute the declarative knowledge of a particular office plan generated through use of the system. Through the analysis of the office building, as discussed above, students construct the structure of the frame. By applying this structure to a specific case, they gain insight in the feedback-loops and recursions that occur when designing an instance of the office building type.

The Knowledge-Based System is to be implemented in AutoLISP. The choice of AutoLISP was made for several reasons: the large amount of experience with both AutoLISP and AutoCAD; the reliable graphic control/output in AutoCAD; and the aim to teach Knowledge-Based Systems programming as well as their use.

Most students do not have extensive programming experience. Therefore, they are first provided with a tutorial on AutoLISP programming. Next, they are shown how a design description can be represented in a frame-like notation, and how an instance can be realised. Through a worked-out example it is demonstrated how a Lispfile is constructed from the analysis of the type (see previous section) and how it is actually programmed. Subsequently, the students will work out a specific subtype and implement it in a Knowledge-Based System.

The sequence the system will pass through develops a design in the first stages of conceptual design. At the end of the last step, the design will be dimensioned, oriented, and positioned. The organisation will be defined through a zoning structure, and specific spaces will be located in the plan. Since the T-Shape is already worked out, students are required to analyse a different subtype of the office building. In the exercise, these are formed by the class of simple shapes: I, T, L, H, +, and \( \therefore \). These shapes can be based on an orthogonal grid, which reduces their geometric complexity, and makes them therefore easier to implement (see Figure 10).
The grid that is used is a hierarchical system of grids of which the modules are whole multiples. This system of grids and rules is referred to as a "generic grid" (Bax 1984). Briefly put, each grid that has a specific module represents an architectural level of scale (e.g., detail level, room level, building level, block level, and urban level). Each level of scale can be designated a specific design participant in the design process (e.g., interior designer on room level, architect on building level, an urban designer on block level; see Figure 11 next page). Through specification how a lower level (smaller module) can fill the next higher level (larger module), it is possible to co-ordinate design participants active in various levels. The concept of the generic grid also designates elements to certain levels of scale. In this way it structures a hierarchy of elements in the system.

The students are provided with a sample Lisp-code of a Knowledge-Based System for the T-Shape which shows how the frame representation is constructed. The frame representation common to all subtypes of the office building is stored separately as a template-file in "office.frm" (see example code in Figure 12; next page). From the template-file the instance frames are constructed in the process of designing an office plan. The slots are ordered in sections. The first section "Office:" contains slots that hold general information: the subtype ("rs_a:"), the required area in the program of demands ("Area:"), the number of stories ("No_stories:"), and the storey-height ("Height_storey:"). In order to obtain a value, the slot calls an AutoLISP-program ("officetype", "def_area", "def_no_stories", and "def_height_story" respectively). A student working on the L-type, for example, must program the appropriate slots and place them in the template-file under the section called "LShape". In this way, the system will extend gradually its command over various subtypes.
The frame in the form used here, is rather limited. Much reasoning is dealt with in the Lisp-code that reads the template-file and makes an instance of an office building. In the exercise this means that students implement the instructions that are mentioned in a slot in a particular sub list of the template-file. For example, each office building has its own specific set of dimensions which are made and stored in a list in the slot named 'Dimensions'.

```
'(
  (Office: (myoffice))
  (Is_a: (officetype))
  (Area: (def_area))
  (No_stories: (def_no_stories))
  (Height_storey: (def_height_story))
)

("TShape"
  (Orientation: (def_T_orient))
  (Dimensions: (def_T_dim))
  (Insertpoint: (def_insert))
)

("LShape"
  ....
)
)
```

Fig. 12 - Sample code of template file "office.frm"
A Students Example

A design layout will be produced by the Lisp-program 'KBSTN . LSP'.
The sequence of questions, answers, remarks and results will be given below.
The questions will be underlined and comments will be in italic.

At first questions will be put about the parcelarea and the area-need for an office building.
Geef lengte Kavel (m): 80
Geef breedte Kavel (m): 80
Totaal behoefte kantooroppervlak: 7500
Er wordt nu een kantoorontwerp gegenereerd!
Geef naam voor het kantoorontwerp: design_01

An orthogonal officetype will be evaluated:

Bepaal orthogonale vorm

Recht
Tvorm
Kruis
Hvorm
Lvorm
Vierkant

Bepaal orthogonale vorm (Recht/Tvorm/Kruis/Hvorm/Lvorm/Vierkant): Kruis

The floorarea and the number of stories will be evaluated from parcelarea and the area-need.
Kies uit het kantooropp: 2*3750(2)/3*2500(3)/4*1875(4)/5*1500 (5)/6*1250(6)/
7*1075(7) 8*940(8): 5

Choose the height of a floor.
Kies de verdiepingshoogte in [m]: 3.0(1)/3.2(2)/3.4(3)/4.0 (4): 3
First estimation \((A=B)\) with the chosen floor area and the question to choose the depth of office-wing.

Eerste schatting (A=B) bij opp. 1500 (m²): 17.32

Kies diepte vleugels / 8-10m (1) / 12-20m (2) / 22-38m (3) / 40m (4):

Definition of the margins at \(A=12\) m and \(A=20\) m.

Bepaling marges bij \(A=12\) m; \(B=28\) m.

Bepaling marges bij \(A=20\) m; \(B=13\) m.
Choose a measure between 13 and 28 m for the length of the wings.  
Kies maat tussen 13 en 28: 20

Dimension values will be given now, also the adapted Area value and the length of 
the office will be given. Afterwards the user will be asked for another measure if 
needed.
Waarden: A=20, B=16, Area=1530 [Lengte=56].
Wilt u toch een andere maat invoeren? (Ja/Nee) <Nee>: Nee

The officetype will be inserted at a given insertion point in the parcel area. 
Geef invoegpunt in tekening: <click in drawing>
Is dit invoegpunt correct (Ja/Nee)? <Ja>: Ja

Result of this process:
Examination of a double zoning if wanted.
Dubbele zonering bekijken? (Ja/Nee) <Ja>: Ja

A suggestion how to divide the zoning with a depth of 16 m.
(dn = depth corridor north-part, dz = depth corridor south part, bpc = width primary circulation)
Kamer kantoor meestal 4.5-6m diep; daglichtdiepte max. 7m; minimumbreedte van primaire circulatie is 2 m.
Voorstel opdeling dubbele zonering bij diepte van 16m:
- 7.0 (dn) - 2.0 (bpc) - 7.0 (dz)
Acordo? (Ja/Nee) <Ja>: Ja

The result of this process:

The final result of the instance of an office-frame:
OFFICE: “design_01”

IS_A: Kruis
AREA: 1536
NO_STORIES: 5
HEIGHT_STOREY: 3.4
ORIENTATION: LL
DIMENSIONS:
A: 16
B: 20
INSERTION_POINT: (22.0 54.0 0.0)
Conclusions

The prestructuring of the analysis through generic representations proved helpful in understanding the reasoning sequence leading to a specific design. It also supported a modular approach in developing the subtypes of the office building. The static structure of the frame was fairly easy defined on the basis of the analysis.

During the course, it appeared that AutoLISP is a rather poor environment for developing Knowledge-Based Systems. Students were required to spend a large amount of time writing relatively unimportant code to support the several routines necessary for implementing the system. In combination with the lack of programming experience this proved to be a bottleneck.

There are two ways of easing the programming-task relative to the exercise. First, by placing the exercise in a more work-intensive context like a project instead of a course. In that way students are able to spend more time working on the implementation. Second, by providing a frame notation that is formulated in more natural language terms. This can result in a less limited frame-representation. The exercise can then concentrate on implementing knowledge in a frame representation technique. The emphasis will be less on programming in AutoLISP. An example of a general syntax for such a frame is provided here:

```lisp
'(Define_General_Concepts
  (SiteWidth
    (IS: Parcel_Width)
    (PROMPT: "Give width of the site: ")
    (VALUE: RANGE: 0 1000)
    (ASK: YES))
  (SiteLength
    (IS: Parcel_Length)
    (PROMPT: "Give length of the site: ")
    (VALUE: RANGE: 0 1000)
    (ASK: YES))
  (SiteSurface
    (IS: Parcel_Area)
    (VALUE: CALC: (* SiteWidth SiteLength))
  )
  (OfficeSurface
    (IS: Office_Area)
    (PROMPT: "Give required floor area: ")
    (VALUE: RANGE: 0 10000)
    (ASK: YES))
  (BasicForm
    (IS: Office_Type)
    (PROMPT: "What platonic shape ")
    (VALUE: CASE: T "T-Shape" H "H-Shape" L "L-Shape" K "K-Shape" R "Square" DEFAULT: T)
    (SHOWSLIDE: "ORTHOTYP")
    (ASK: YES))
  (StoreyHeight
    (IS: Office_Height_Of_Storys)
    (PROMPT: "Give the storey height: ")
    (VALUE: CASE: 1 "3.00 m" 2 "3.10 m" 3 "3.40 m"
    4 "3.70 m" 5 "4.20 m" DEFAULT: 3)
    (ASK: YES))
  (No_Stories
    (IS: Office_No_of_Storys)
    (VALUE: FROM: Floorarea)
  )
  (Floorarea
    (IS: Office_Floor_Area)
    (VALUE: WHILE
      DEP_OF: OfficeSurface
      COND: (< No_Stories 8)
      (=> Floorarea 800))
    (ASK: NOT))
  )
  (T
    (Orientation
      (IS: Office_Orientation)
      (PROMPT: "Give orientation: ")
      (VALUE: CASE: LL RR LU RD)
      (SHOWSLIDE: "T-SLIDE")
      (ASK: YES))
  )
  (Dimensions
    (IS: Office_Dimensions)
    (VALUE: DIM_LIST:
      DIM_COND: (= A E C)
      (= B D))
    (SHOWSLIDE: "SLIDE")
    (ASK: YES))
  )
  (InsertionPoint
    (IS: Office_InsPnt)
    (VALUE: BECALC)
  )
)
Students reported that they had gained insight into the office building type with which most of them were not very familiar beforehand. The information provided in the knowledge base showed the declarative content of the type, and demonstrated how knowledge could be extracted from architectural sources. Knowledge acquisition aspects related to Knowledge-Based Systems were introduced through selecting those pieces of information necessary for developing the subtypes. Some information in the prestructured material provided for, was found redundant, and some inconsistencies between sources were detected. Basically, students followed the worked out example of the analysis and re-used the knowledge provided there.

Through applying the knowledge in the analytical framework provided for by generic representations, students learned about the procedural aspects of a design process. They saw the role declarative knowledge has in this process. In this way they also gained insight into design modelling of a building type. In conclusion, the students learned about the significance of typological knowledge in architectural design and the design process.

References


Bax, M.F.Th., The Design of a Generic Grid. 1984. Open House, vol. 9, no. 4


Listings of Lisp-files

Template file of an office; the slots will be evaluated by a call of the corresponding action.

```lisp
;; FILE: OFFICE.FRM
 Office: (myoffice)
 (Is_a: (officetype))
 (Area: (def_area))
 (No_stories: (def_no_stories))
 (Height_storey: (def_height_storey))

 (*Tvorm*
  (Orientation: (def_T_orient))
  (Dimensions: (def_T_dim))
  (Insertpoint: (def_insert))
)

 (*Kruis*
  (Orientation: (def_Kruis_orient))
  (Dimensions: (def_Kruis_dim))
  (Insertpoint: (def_insert))
)

KBSTNLSP: the Lisp-file with global variables definitions and the activation of the design process

```
** Settings
(setvar "DIMTXT" 1.2) ; texthoogte binnen dimensionlines
(setvar "DIMASZ" 1) ; arrowsize dimensionlines
(setvar "DIMES1" 1) ; geen extensionline bij punt1
(setvar "DIMES2" 1) ; geen extensionline bij punt2
(setvar "SNAPUNIT" '(1 1))
(setvar "PDMODE" 3)
(setvar "PDSIZE" 1)
(command "LAYER" "M" "dimensions" "")
(command "LAYER" "M" "zonering_dubbel" "")
(command "LAYER" "M" "kavelomtrek" "")
*** Overige settings
(command "units" 2 "" "")
(command "LIMITS" "0,0"
(strcat (itoa (+ Kavel_lengte 25)) "," (itoa (+ Kavel_breedte 25))))
(command "ZOOM" "A")
(command "LAYER" "M" "kavel" "")
)

;*******************************************************************************
;** Een kantoorontwerp genereren
(setq stop nil)
(while (null stop)
(redraw)
(princ "\nEr wordt nu een
(setq Newframe nil)
(make_instance 'Office;) ; *** Platonische vorm ***
(cond (= Offtype "Tvorm") (make_instance '"Tvorm")
((= Offtype "Kruis") (make_instance '"Kruis")
)
(initget "J Ha N Nee")
(setq antw (getkword "\nDubbele zonering zonder contour bekijken? (Ja/Nee) <Ja>: "))
(if (or (= antw "Ja") (= antw "J") (null antw))
(progn
   (cond (= Offtype "Tvorm") (zone_T_dub_zc)
   ((= Offtype "Kruis") (zone_Kruis_dub_zc))
)
)
(show_frame Newframe)
(initget "J Ha N Nee")
(setq antw (getkword "\ndoorgaan met een nieuw ontwerp (Ja/Nee) <Nee>: "))
(if (or (= antw "Ja") (= antw "J")
(progn
   (setq stop nil)
   (princ "\nDoorgaan met een nieuw ontwerp")
   (command "ERASE" "C" '(0 0 0)
     (list (+ Kavel_lengte 20) (+ Kavel_breedte 20) 0) "")
)
(progn
   (setq stop T)
   (princ "\nOntwerp in een Casebase opslaan!")
   (terpri)
   ;(command "QUIT" "y")
)
)

;*******************************************************************************
;** Eind kantoorontwerp genereren
;*******************************************************************************

OFFICETNLSP: Lisp-file with the actions to evaluate the slots of different officetypes

;*******************************************************************************
;*** FILE: OFFICETNL.LSP
;*******************************************************************************

;*******************************************************************************
;*** SECTION: Procedures t.b.v. de Frame-definities
;*******************************************************************************

;*******************************************************************************
;*** Framename definition
(defun myoffice ()
  (getstring "Geef naam voor het kantoorontwerp: ")
)

;*******************************************************************************
;*** End of Framename definition
(defun officetype ( / antw)
(command "VSLIDE" "A:ORTHOTfn")
(initget 1 "R Recht T Tvorm K Kruis H Hvorm L Lvorm V Vierkant")
(setq antw
  (getkword "Kies orthogonale vorm (Recht/Tvorm/Kruis/Hvorm/Lvorm/Vierkant:")
)
(cond ((or (= antw "Recht") (= antw *R*)) (setq Offtype "Recht"))
  ((or (= antw "Tvorm") (= antw *T*)) (setq Offtype "Tvorm"))
  ((or (= antw "Kruis") (= antw *K*)) (setq Offtype "Kruis"))
  ((or (= antw "Hvorm") (= antw *H*)) (setq Offtype "Hvorm"))
  ((or (= antw "Lvorm") (= antw *L*)) (setq Offtype "Lvorm"))
  ((or (= antw "Vierkant") (= antw *V*)) (setq Offtype "Vierkant"))
)
(redraw)
Offtype
)
;*** End of Define office type ***

;*** Define area of a storeyfloor and number of stories ***
(defun def_area (/ stories opp stop initstr antwstr lstories lopp)
  (setq stories 1
    opp Totalopp
    stop nil
    initstr ""
    antwstr "Kies uit het kantooropp:"
    lstories nil
    lopp nil)
  (while (null stop)
    (if (not (< opp Totalopp opp))
      (setq initstr (strcat initstr (itoa stories) " ft" (itoa stories) "*" (itoa opp) "")
         antwstr (strcat antwstr (itoa stories) "*" (itoa opp)
         "*" (itoa stories) ")")
         lstories (append lstories (list stories))
         lopp (append lopp (list opp))
    )
    (setq opp (1+ opp))
    (if (not (= opp 800)) (setq stop T))
  )
  (setq stories (1+ stories)
    opp (/ Totalopp stories))
  (if (not (= (rem opp 10) 0))
    (while (not (= (rem opp 5) 0)) (setq opp (1+ opp)))
  )
  (if (or (> stories 8) (< opp 800)) (setq stop T))
  (setq antwstr (strcat (substr antwstr 1 (1- (strlen antwstr))) ":"))
  (initget (+ 1 2 4) initstr)
  (setq antw (getkword antwstr))
  (setq No_stories (atoi antw))
  (setq Area (nth (- No_stories (- (last lstories) (length lstories)) 1) lopp))
)
;*** End of Define area of a storeyfloor and number of stories ***

;*** Define number of stories ***
(defun def_no_stories () No_stories)
;*** End of Define number of stories ***

;*** Define height of a storey ***
(defun def_height_storey ()
  (initget "1 2 3 4")
  (princ "Kies de verdiepingshoogte in [m]:")
  (setq antw (getkword "n 3.0 (1) / 3.2 (2) / 3.4 (3) / 4.0 (4) :"))
  (cond ((= antw "1") (setq height_storey 3.0))
    ((= antw "2") (setq height_storey 3.2))
    ((= antw "3") (setq height_storey 3.4))
    ((= antw "4") (setq height_storey 4.0))
  )
)
;*** End of Define height of a storey ***

;*** Define Tvorm orientation ***
(defun def_T_orient (/ initstr antw)
  (command "VSLIDE" "A:TSLIDE")
  (initget "LL LU RR RD")
  (setq antw (getkword "Kies de verdiepingshoogte in [m]:")
  )
)

(setq Orient antw)
)
;*** End of Define Tvorm orientation ***
;
;*** Define Kruis-vorm orientation ***
(defun def_Kruis_orient (/ antw)
  (setq antw "LL")
  (setq Orient antw)
)
;*** End of Define Kruis-vorm orientation ***
;
;*** Define Tvorm dimensions ***
(defun def_T_dim (/ a b c d e sl antw dmin dmax kb k1 maat1 maat2 maat stop opp 1engte)
  (command "VSLIDE" (strcat "T" Orient "SLIDE"))
  (setq sl (sqrt (/ Area 4)))
  (princ (strcat "NBedoelde schatting (A=B+C=D=E) bij opp. " (itoa area) " (m2): " (rtos sl 2 2))
  (initget "1 2 3 4")
  (setq antw (getkword "Kies diepte vleugels / 8-10m (1)/ 12-20m (2)/ 22-38m (3)/ 40m (4): "))
  (cond ((= antw "1") (setq dmin 8 dmax 10))
        ((= antw "2") (setq dmin 12 dmax 20))
        ((= antw "3") (setq dmin 22 dmax 38))
        ((= antw "4") (setq dmin 40 dmax 40))
)
  (setq kb Kavel_breedte kl Kavel_lengte)
  (setq Stpnt
    (cond ((= Orient "LL") (list 10 (+ (/ kb 2) 10) 0))
          ((= Orient "LU") (list (+ (/ k1 2) 20) (+ kb 10) 0))
          ((= Orient "RR") (list (+ k1 10) (+ (/ kb 2) 10) 0))
          ((= Orient "RD") (list (+ (/ k1 2) 10) kb 10) 0))
    )
  (setq maat2 (/ (/ (- Area (* dmin dmin)) 3) dmin)
          maat1 (/ (/ (- Area (* dmax dmax)) 3) dmax))
  (princ (strcat "NBedoelde marges bij B=D=" (itoa dmin) "m ; "
             "A=E=C=" (itoa maat2) "m."))
  (redraw)
  (draw_Tvorm T Stpnt Orient maat2 dmin maat2 dmin maat2)
  ;*** Gespecificeerde vorm ***
  (command "DELAY" 3000)
  (if (/= dmin dmax)
    (progn
      (entdel Drawent)
      (del_dim Diment)
      (princ (strcat "NBedoeling marges bij B=D=" (itoa dmax) "m ; "
               "A=E=C=" (itoa maat2) "m."))
      (draw_Tvorm T Stpnt Orient maat1 dmax1 maat2 dmax1)
    )
  )
  (setq stop nil)
  (while (null stop)
    (setq maat 0)
    (if (= maat dmin)
      (progn
        (princ "NVoork de diepte is 40m gekozen!")
        (setq d 40)
      )
    )
    (while (or (< maat maat1) (> maat maat2))
      (setq maat (getint (strcat "Nkies maat tussen " (itoa maat1) " en " (itoa maat2) " : "))
    )
  )
  (setq d dmin opp 0)
  (while (< opp Area)
    (setq lengte (+ (* 2 maat) d)
      opp (+ (* d d) (* 3 maat d))
    d (1+ d)
  )
  (setq d (1- d))
  (entdel Drawent)
  (del_dim Diment)
  )
(draw_Tvorm T Stpnt Orient maat d maat d maat)
(cond (and (= (rem d 2) 0) (= (rem d 5) 0)
(setq "J" Ja N Nee")
(setq antw (getkword (strcat \"Wilt u de diepte op een veelvoud van 5 geeft overschrijding van het \" (itoa Area) "m2 opp." " (Ja/Nee) <Nee>: \")
(if (or (= antw "J") (= antw "Ja")
(while (/= (rem d 5) 0) (setq d (1+ d) lengte (+ (* 2 maat) d)))
)
)
((/= (rem d 5) 0)
(setq antw (getkword (strcat \"Wilt u de diepte op een veelvoud van 2, 5 of geen veelvoud (geeft overschrijding van het \" (itoa Area) "m2 opp.) (2/5/Geen)<Geen>: \")
(cond (<= antw 2) (setq d (1+ d) lengte (+ (* 2 maat) d))
(= antw 5) (while (/= (rem d 5) 0) (setq d (1+ d) lengte (+ (* 2 maat) d)))
)
)
)
(setq opp (+ (* d d) (* 3 maat d)))
(princ (strcat \"Bepaling marge bij opp. * (itoa Area) m2 opp.): \")
(setq a maat c a e a b d)
(redraw)
(princ \"\")
(entdel Drawent)
(del_dim Diment)
(draw_Tvorm T Stpnt Orient a b c d e)
(setq "J" Ja N Nee")
(setq antw (getkword \"Wilt u toeh een andere maat invoeren? (Ja/Nee) <Nee>: \")
(setq stop (if (or (= antw "Ja") (= antw "J") nil) T))
)
(redraw)
(setq Area opp)
(setq Newframe (append (list (car Newframe) (cdr Newframe))
(subst (list 'AREA: Area) (assoc 'AREA: (cdr (cdr Newframe)))))
(setq Diment (list a b c d e))
)
;*** End of Define Tvorm dimensions ***

;*** Define Kruis-vorm dimensions ***
(defun def_Kruis_dim (/ a b c sl antw dmin dmax kb k1 maat1 maat2 stop opp lengte)
(command "VSLIDE" "a:kruis1")
(setq sl (sqrt (itoa Area 5)))
(princ (strcat \"Eerste schatting (A=B) bij opp. * (itoa area) \" (itoa sl 2 2) \")
(=sl sl 2 2)
(=sl sl 2 4)

(setq antw (getkword \"Kies diepte vleugels / 8-10m (1)/ 12-20m (2)/ 22-38m (3)/ 40m (4): \")
(cond (<= antw "1") (setq dmin 8 dmax 10))
(= antw "2") (setq dmin 12 dmax 20)))
(= antw "3") (setq dmin 22 dmax 38))
(= antw "4") (setq dmin 40 dmax 40))

(setq kb Kaveel_breedte kl Kaveel_lengte)
(setq Stpnt)
(cond (<= Orient "LL") (list 10 (+ (/ kb 2) 10) 0)
(<= Orient "LV") (list (+ (/ kl 2) 20) (+ kb 10) 0)
(= Orient "RR") (list (+ kl 10) (+ (/ kb 2) 10) 0))
(= Orient "RD") (list (+ (/ kl 2) 10) 10 0))
)

(setq maat2 (/ (/ (/ Area (+ dmin dmin)) 4) dmin)
maat1 (/ (/ (/ Area (+ dmax dmax)) 4) dmax)
(princ (strcat \"Bepaling marge bij A=\" (itoa dmin) "m; \"B=\" (itoa maat2) "m;\")
(redraw)
(draw Kruis T Stpnt Orient dmin maat2)
;*** Gespecificeerde vorm ***
(command "DELAY" 3000)
;;; Define insertion point kantoorgebouw ***
(defun def_insert (/ stop antw insent)
  (setq stop nil)
  (command "LAYER" "S" "kavelomtrek")
  (setq PLINEWID 0.4)
  (command "PLINE" '(0 0 0) (list (+ 30 Kavel_lengte) 0 0)
    (list (+ 30 Kavel_lengte) (+ 30 Kavel_breedte) 0)
    (list 0 (+ 30 Kavel_breedte) "C")
  (command "LAYERS" "S" "kavel"
  (command "INSERT" "a:orient" (list (+ 10 Kavel_lengte) 10 0) 0.2
  (setvar "GRIDMODE" 1)
  (setvar "SNAPMODE" 1)
  (while (null stop)
    (command "POINT" Stpnt)
    (setq insent (getpoint Stpnt "Geef invoegpunt in tekening: ")
    (enddel Drawent)
    (enddel insert)
    (command "POINT" Ins_pnt)
    (setq insent (getpoint Ins_pnt "Geef invoegpunt in tekening: ")
    (enddel insert)
    (cond ((= Offtype "Tvorm")
      (draw_Tvorm T Ins_pnt Orient (car Dimlst) (cadr Dimlst)
        (caddr Dimlst) (cadddr Dimlst) (last Dimlst))
    )
    (= Offtype "Kruis")
    (draw_Kruis T Ins_pnt Orient (car Dimlst) (last Dimlst))
    (T princ "Andere vormen nog niet geïmplementeerd!")
    )
  )
  (initget "J Ja N Nee")
  (setq antw (getkword "Is dit invoegpunt correct (Ja/Nee)? <Ja>: ")
  (if (or (= antw "N") (= antw "Nee")
    (progn (setq Stpnt Ins_pnt) (enddel Insent)
    (progn
      (setq stop T Drawent)
      (cond ((= Offtype "Tvorm")
        (draw_Tvorm nil Ins_pnt Orient (car Dimlst) (cadr Dimlst)
          (caddr Dimlst) (cadddr Dimlst) (last Dimlst))
        (enddel insert)
      )
      (= Offtype "Kruis")
      (draw_Kruis T Ins_pnt Orient (car Dimlst) (last Dimlst))
      (enddel insert)
    )
    (T princ "Andere vormen nog niet geïmplementeerd!")
    )
  )
)
(setvar "GRIDMODE" 0)
ins_pnt

;;; End of Define insertion point kantoorgebouw ***

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; SECTION: Hulp procedures t.b.v. Procedures m.b.t. Frame-definities ***
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; Make instance of template frame ***

;;; based on work of Marc v. Groote and Marco Vlemmix (ICAD '93/'94) ***
(defun make_instance (ftype / fdef fobj slotnr slotl)
  (setq fdef (assoc ftype (load "a:office.frm")))
  (setq fobj (list (car fdef))
    slotnr 1
  (if (null Newframe)
    (setq Newframe (append fobj (list (eval (cadr fdef))))
    slotnr 2)
  )
  (while (not (null (setq fslot (nth slotnr fdef))))
    (setq slotnr (1+ slotnr))
    (setq slot (list (car fslot))
    (setq slot (append slot (list (eval (cadr fslot)))))))
(setq Newframe (append Newframe (list slotl)))

;;;; End of Make instance of template frame

;;;; Draw Tvorm

(defun draw_tvorm (flag p0 o a b c d e /
  xp0 yp0 zp0 diml l l insert)
  (setq xp (car p0) yp (cadr p0) zp (last p0) l (+ a b c))
  (cond ((= o "LL")
    (setq xp (+ xp a) p1 (list xp yp zp)
      yp (+ yp a) p2 (list xp yp zp)
      xp (+ xp b) p3 (list xp yp zp)
      yp (+ yp b) p4 (list xp yp zp)
      xp (+ xp c) p5 (list xp yp zp)
      yp (+ yp c) p6 (list xp yp zp)
      xp (+ xp l) p7 (list xp yp zp)
      diml (list "@0,-2" "@-2,0" "@0,-2" "@2,0"
              "@0,-2" "@2,0" "@0,2" "@-2,0")
    )
    ((= o "RR")
      (setq xp (- xp a) p1 (list xp yp zp)
        yp (- yp a) p2 (list xp yp zp)
        xp (- xp b) p3 (list xp yp zp)
        yp (- yp b) p4 (list xp yp zp)
        xp (- xp c) p5 (list xp yp zp)
        yp (- yp c) p6 (list xp yp zp)
        xp (- xp l) p7 (list xp yp zp)
        diml (list "@2,0" "@0,2" "@-2,0" "@0,-2"
              "@2,0" "@-2,0" "@0,2" "@-2,0")
    )
    ((= o "LU")
      (setq xp (xp a) p1 (list xp yp zp)
        yp (+ yp a) p2 (list xp yp zp)
        xp (+ xp b) p3 (list xp yp zp)
        yp (+ yp b) p4 (list xp yp zp)
        xp (+ xp c) p5 (list xp yp zp)
        yp (+ yp c) p6 (list xp yp zp)
        xp (+ xp l) p7 (list xp yp zp)
        diml (list "@-2,0" "@0,2" "@-2,0" "@0,-2"
              "@2,0" "@0,2" "@-2,0" "@0,-2")
    )
    ((= o "RD")
    (setq xp (+ xp a) p1 (list xp yp zp)
      yp (+ yp a) p2 (list xp yp zp)
      xp (+ xp b) p3 (list xp yp zp)
      yp (+ yp b) p4 (list xp yp zp)
      xp (+ xp c) p5 (list xp yp zp)
      yp (+ yp c) p6 (list xp yp zp)
      xp (+ xp l) p7 (list xp yp zp)
      diml (list "@2,0" "@0,-2" "@2,0" "@0,2"
              "@2,0" "@0,2" "@-2,0" "@0,-2")
    )
    )
  )
  (if flag
    (progn
      (setq *PLINEMID* 0.2)
      (command "PLINE" p0 p1 p2 p3 p4 p5 p7 "c")
      (setq "PLINEMID" 0)
      (setq Drawent (entlast))
    )
  )
  (command "LAYER" "S" "dimensions" "")
  (command "DIM" "ALIGNED" p0 p1 (nth 0 diml) a)
  (command "DIM" "ALIGNED" p1 p2 (nth 1 diml) a)
  (command "DIM" "ALIGNED" p2 p3 (nth 2 diml) b)
  (command "DIM" "ALIGNED" p3 p4 (nth 3 diml) a)
  (command "DIM" "ALIGNED" p4 p5 (nth 4 diml) c)
  (command "DIM" "ALIGNED" p5 p6 (nth 5 diml) d)
  (command "DIM" "ALIGNED" p6 p7 (nth 6 diml) l)
  (command "DIM" "ALIGNED" p7 p0 (nth 7 diml) d)
  (command "DIM" "EXIT")
  (setq Diment (SSGET "X" (list (cons 8 "dimensions"))))
  (command "LAYER" "S" "kavel" "")
)
(defun draw_kruis (flag o a b / xp0 yp0 zp0 dim1 i insent)
  (setq xp (car pO) yp (cadr pO) zp (last p0) )
  (setq xp (+ xp b) p1 (list xp yp zp)
    yp (- yp b) p2 (list xp yp zp)
    xp (+ xp a) p3 (list xp yp zp)
    yp (+ yp a) p4 (list xp yp zp)
    xp (+ xp b) p5 (list xp yp zp)
    yp (+ yp b) p6 (list xp yp zp)
    xp (- xp b) p7 (list xp yp zp)
    yp (- yp a) p8 (list xp yp zp)
    xp (- xp a) p9 (list xp yp zp)
    yp (+ yp b) p10 (list xp yp zp)
    xp (- xp b) p11 (list xp yp zp)
  
  dim1 (list "@0,-2" "@0,-2" "@2,0" 
    "@0,-2" "@2,0" "@0,2" "@2,0" 
    "@-2,0" "@0,2" "@-2,0")
  )

(if flag
  (progn
    (setvar "PLINEWID" 0.2)
    (command "PLINE" pO p1 p2 p3 p4 p5 p6 p7 p8 p9 p10 p11 "c")
    (setvar "PLINEWID" 0)
    (setq Drawent (entlast»
  )))
)

(command "LAYER" "S" "dimensions" "
(command "DIM" "ALIGNED" p0 p1 (nth 0 dim1) b)
(command "DIM" "ALIGNED" p1 p2 (nth 1 dim1) b)
(command "DIM" "ALIGNED" p2 p3 (nth 2 dim1) a)
(command "DIM" "ALIGNED" p3 p4 (nth 3 dim1) b)
(command "DIM" "ALIGNED" p4 p5 (nth 4 dim1) b)
(command "DIM" "ALIGNED" p5 p6 (nth 5 dim1) a)
(command "DIM" "ALIGNED" p6 p7 (nth 6 dim1) b)
(command "DIM" "ALIGNED" p7 p8 (nth 7 dim1) b)
(command "DIM" "ALIGNED" p8 p9 (nth 8 dim1) a)
(command "DIM" "ALIGNED" p9 p10 (nth 9 dim1) b)
(command "DIM" "ALIGNED" p10 p11 (nth 10 dim1) b)
(command "DIM" "ALIGNED" p11 p0 (nth 11 dim1) a)
(command "DIM" "EXIT")

(setq Diment (SSGET ·X· (list (cons 8 "dimensions")[

(command "LAYER" "S" "kavel" "

;*** Delete dimensions selection ***
(defun del_dim (sel / 1 i)
  (setq 1 (sslength sel) i 0)
  (while (< i 1) (entdel (ssname sel i)) (setq i (1+ i))
    (redraw)
  ))

;*** End of Delete dimensions selections ***
;

;*** Show frame ***
(defun show_frame (frame / i j ch len dl allslots slot slotname slotvalue)
  (princ (car frame)) (princ "\t") (prinl (cadr frame))
  (setq allslots (cdr (cdr frame))
    len (length allslots)
    i 0)
  (while (< i len)
    (setq slot (nth i allslots)
      slotname (car slot)
      slotvalue (cadr slot)
      (princ "\n\t") (princ slotname)
      (cond ((= slotname 'DIMENSIONS:)
        (setq dl (length slotvalue) ch 65 j 0)
        (while (< j dl)
          (princ (read (strcat (chr ch) #: "
          ) (princ (nth j slotvalue))
          (setq j (1+ j) ch (1+ ch))
        ))
      )
    )
  )
)
TUE / Building Information Technology

;*** End of Show frame ***

;*** Dubbele zone representatie zonder contour T-kantoorvorm ***
(defun zone_T_dub_sc (/ a b c d e dn ds bpc abc de antw nil)
  ; d: diepte
  ; dn: diepte noordkant
  ; ds: diepte zuidkant
  ; bpc: breedte primaire circulatie
  ; abc: a+b+c
  ; de: d+ e
  (setq a (car Dimlst) b (cadr Dimlst) e (caddr Dimlst)
       d (cadddr Dimlst) e (last Dimlst)
       abc (+ a b c) de (+ de e))
  (command "LAYER" "S" "zonering_dubbel" "")
  (princ (strcat "a:repzonc" "4,2,0. It..")
  (setq bpc 2.0
       dn (if (> dn 7) (setq bpc 3.0
                       dn (- (- d bpc) 2) dz dn)))
  (if (or (= antw "Nee") (= antw "NU")
         (progn
           (setq atop nil)
           (while (null atop)
             (princ (strcat "Geef
                          achtereenvolgens nieuwe waarden voor
                          dn, bpc en dzl")
             (setq dn (getreal (strcat "Geef
                           waarde voor dn (diepte
                           1 van dubbele zone): ")
               bpc (getreal (strcat "Geef
                              waarde voor bpc
                              (primaire circulatie): ")
               dz (getreal (strcat "Geef
                                waarde voor dz
                                (diepte 2 van dubbele
                                zone): ")
               (if (/= d (fix (+ dn bpc dz)))
                 (princ "Ingevoerde waarden niet correct! Opgenieuw doen!")
                 (setq stop T))
               )))
  (command "LAYER" "S" "zonering_dubbel" "")
  (command "INSERT" "a:repzonct" "4,2,0. It..")
  (command "LAYER" "OFF" "dimensions,kavel" "")
  (setq left (- (car Ins_pnt) 2) right (+ left abc 4)
       down (- (cadr Ins_pnt) e 2) up (+ down de 4))
  (cond ("Orient "L")
    (setq p1 (list left (setq hl (+ (cadr Ins_pnt) (/ dz 2))) 0)
       p2 Ins_pnt
       p3 (list (+ (car Ins_pnt) abc) (cadr Ins_pnt) 0)
       p4 (list right hl 0))
    (command "PLINE" p1 p2 p3 p4 "")
    (command "MIRROR" "L" "p1 p4")
    (setq p1 (list left (setq h1 (+ (cadr Ins_pnt) (/ dn 2) ds bpc)) 0)
       p2 (list (cadr Ins_pnt) (setq h2 (+ (cadr Ins_pnt) dz bpc)) 0)
       p3 (list (+ (car Ins_pnt) abc) h2 0)
       p4 (list right hl 0))
    (command "PLINE" p1 p2 p3 p4 "")
    (command "MIRROR" "L" "p1 p4")
    (setq p1 (list (cadr Ins_pnt) (cadr Ins_pnt) (last Dimlst)) 0)
    (setq p1 (list (setq hl (+ (car ip) (/ dz 2))) 0)
       p2 ip
       p3 (list (car ip) (+ (cadr ip) de) 0)
       p4 (list hl up 0))
    (command "PLINE" p1 p2 p3 p4 "")
    (command "MIRROR" "L" "p1 p4")
    (setq p1 (list (+ (car ip) bpc dz) (cadr ip) 0))
    (setq p1 (list (setq hl (+ (car ip) (/ dn 2))) 0)
       p2 ip)
  )
)

30

BIT Notes
TIlE

Building Information Technology

p3 (list (car ip) (+ (cadr ip) de) 0)
p4 (list h1 up 0)
(command "PLINE" pi p2 p3 p4 **)
(command "MIRROR" "L" ** pi p4 **)

(= Orient "RR")

(= Orient "LU")

(= Orient "RD")

(redraw)
(command "ZOOM" "ALL")
(command "DELAY" 4000)
(command "LAYER" "ON" "dimensions,kavel" **)
(command "DELAY" 4000)

;;*** End of Dubbele zone representatie zonder contour T-kantoorvorm ***

;;*** Dubbele zone representatie zonder contour Kruis-kantoorvorm ***
(defun zone_Kruis_dub_zc (a b dn dz bpc abb antw nil)
a: diepte
dn: diepte noordkant
dz: diepte zuidkant
bpc: breedte primaire circulatie
abb: a+b+b

(setq a (car Dimlst) b (last Dimlst) abb (+ a b b))

(command "LAYER" "S" .zonering_dubbe1 11 n.)

(princ (strcat 

Kamerkantoor meestal 4.5-6 m diepi
daglichtdiepte max. 7m;
MinimUlllbreedte van primaire circulatie is 2m.))

(setq bpc 2.0 dz dn)
(if (> dn 7) (setq bpc 3.0 dz dn))

(= Orient "LL")

(setq pi (list left (setq hl (+ (cadr Ins_pnt) (/ dn 2) 0)) 0)
p2 Ins_pnt
p3 (list (+ (cadr ins_pnt) 0) (cadr ins_pnt) 0)
p4 (list right h1 0))

(command "PLINE" pi p2 p3 p4 **)
(command "MIRROR" "L" ** pi p4 **)

(setq pl (list left (setq h1 (+ (cadr ins_pnt) (/ dz 2))) 0)
p2 Ins_pnt
p3 (list (+ (cadr ins_pnt) 0) (cadr ins_pnt) 0)
p4 (list right h1 0))

(command "PLINE" pi p2 p3 p4 **)
(command "MIRROR" "L" ** pi p4 **)

(setq ip (list (+ (cadr ins_pnt) 0) (cadr ins_pnt) 0)
(- (cadr ins_pnt) (last Dimlst) 0))

(setq pl (list (setq h1 (+ (cadr ip) (/ 2 dz 2))) down 0)
p2 ip
p3 (list (car ip) (+ (cadr ip) abb) 0)
p4 (list hl up 0))
(command "PLINE" pl p2 p3 p4 **)
(command "MIRROR" "L" ** pl p4 **)
(setq ip (list (+ (car ip) bpc dz) (cadr ip) 0))
(setq pl (list (setq hl (+ (car ip) (/ dn 2))) down 0)
p2 ip
p3 (list (car ip) (+ (cadr ip) abb) 0)
p4 (list hl up 0))
(command "PLINE" pl p2 p3 p4 **)
(command "MIRROR" "L" ** pl p4 **)
)
(redraw)
(command "ZOOM" "ALL")
(command "DELAY" 4000)
(command "LAYER" "ON" "dimensions,kavel" **)
(command "DELAY" 4000)
)
;*** End of Dubbele zone representatie zonder contour Kruis-kantoorvorm ***