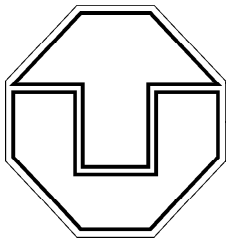
Assistance in Conceptual Design of Concrete Structures by a Description Logic Planner



Genehmigte Dissertation
von der Fakultät für Bauingenieurwesen
der Technischen Universität Dresden
zur Erlangung des akademischen Grades eines Doktors
der Ingenieurwissenschaften

vorgelegt von
Dipl.-Ing., MSc Michael Eisfeld aus Kassel

Gutachter: Universitätsprofessor Dr.-Ing. Raimar J. Scherer
Universitätsprofessor Dr.-Ing. Franz Baader
Universitätsprofessor Dr. habil. Peter Struss

Preface

> "To design a structure and give it the correct proportions, one needs to follow two paths: the intuitive and the mathematical one."
>
> (Pier Luigi Nervi, 1955)

This thesis is the result of my research work at the Institute of Construction Informatics at the University of Dresden from August 2000 to December 2003. The work about computer-aided conceptual design of building structures was initiated by Professor Raimar Scherer. He investigated research into this area at the institute with the Doctoral thesis of Marcus Hauser in 1994 under his supervision. I continued and expanded research into this challenging field under his supervision as part of the European Industrial Research and Development Project ISTForCE. My thesis has given me the chance to explore Navi's statement that is at the heart of structural design despite the fact that it has not been the guiding principal of mainstream applied informatics that deals with problems of structural design.

The development of the computer-aided design system in this thesis is based on a range of design studies and interviews. They were conducted at Ove ARUP, Dusseldorf [Eis01a], Jacobsen&Widmark, Gothenburg [Eis00], and at the design office of Eisfeld Engineers, Kassel (published in this thesis).

This thesis represents original work by the author and has not been submitted in this or any other form. Where use has been made of the work of others, it has been duly acknowledged in the text.

Acknowledgments

I have been very fortunate to receive the support of many people who contributed to this thesis in numerous ways.

My parents have been a constant source of support. Therefore and for the many times my father and I discussed the "secrets" of good conceptual design of concrete structures I would like to thank them.

I am grateful to Dr-Ing. Raimar J. Scherer, Dr-Ing. Franz Baader, and Dr. habil. Peter Struss for their guidance.

I would like to thank Dr. Till Eggers and Marcus Looft in particular for valuable and detailed reviews on my thesis. Their kind and careful reviews made this thesis a better thesis.

I have extensively benefited from correspondences with several computer scientists, which was very helpful for me as a structural engineer. I would like to thank Dr. habil. Ralf Möller, Dr. Carsten Lutz for their help on description logics, Katharina Wolter for her comments on the GUI design of the system, and Michael Wessel for comments on my journal paper.

I would also like to acknowledge the work of CogVis group of the University of Hamburg and Planning group of the University of Maryland, which provided the description logic system RACER and planning system SHOP used in the thesis.

This thesis depends on information covered by design studies at Ove ARUP, Jacobsen&Widmark Consulting, and Eisfeld Engineers. I would like to thank these companies for making detailed design studies possible.

I would also like to thank my office mate Alexander Gehre who introduced me into the basics of object-oriented programming.

My wife, Sabine Sors-Eisfeld, gave me the necessary personal support over the years of my thesis. If she had not taken the decision to follow me to Dresden, I probably would have not had the possibility of doing this thesis.

I dedicate this thesis to her.

Table of Content

List of Figures

Abstract

Conceptual design of concrete structures requires detailed knowledge. The lack of such knowledge results in ill-designed *structural concepts*, which have to be corrected or revised in a "time consuming" iterative process. One way to improve the outcome of the design process is to assist engineers with *computer-aided design systems* that make knowledge of experienced practitioners usable. However, todays computer-aided design systems miss the opportunity to assist effectively.

Expressive description logics appear to be an appropriate tool for conceptual design support because they allow the formal representation of and reasoning about incomplete *conceptual knowledge* in a correct way. Knowledge representation systems like RACER enable description logics effective processing such as *consistency* and *completeness testing* of a structural concept during conceptual design on a predefined knowledge base. Description logics extended by constructs for actions and plans can be additionally employed for specifying *control knowledge* by which practitioners *hierarchically plan* their design process to reduce later iterations and subsequent costs.

The result of this thesis is an interactive description logic based planning formalism. By the formalism, engineers can draw up robust structural concepts closely related to their mental model, test them for correctness on the knowledge base, plan thereby their design process, and reduce time consuming iterations. To reach this result, the conceptual design task of concrete building structures has been formalized, and design studies of experienced engineers have been analyzed to acquire the necessary information about their reasoning and employed knowledge. As a proof of concept, a prototype system based on the formalism has been implemented.

Keywords: computer-aided design, CAE, interactive description logic planning, conceptual structural design, declarative control knowledge, design problem solving, configuration design

Chapter 1

Introduction

The thesis develops a theoretical framework and a practical approach that supports structural engineers by computer-aided conceptual design. After the structural design process has been introduced briefly and an example has clarified the problems engineers face at the conceptual design stage, I argue that a conceptual model about the structure is of primary importance to design a correct structural concept and to reduce design iterations by control knowledge. Commercial tools and various research approaches proposed to date do not support the use of incomplete and abstract representations of the design structure in form of such a model. This thesis attempts to fill the gap in computer-aided conceptual design of concrete building structures by an interactive hybrid system based on description logic and hierarchical planning.

1.1 The Design Process

Various tasks have to be fulfilled during the building design process to ensure the main building functions of occupancy, load transfer, and services [BD95]. One possibility of dividing the design process into phases is shown in figure 1.1 for which the structural design phases are depicted in grew and the iteration cycles are omitted for clarity. The first phase consists of the conceptual design of the architect. At this stage, the architect mainly focuses on the appearance and the occupancy of the building with its particular rooms. The result is a *room and occupancy concept*, which determines the geometry and occupancy loads for a building structure. In the next step of the design process, the architect or the structural engineer designs a *structural configuration* that exhibits a global *load transfer*. This task results in the main arrangement and choice of types of *structural elements*, which are ordinarily columns, walls,

slabs, beams, shear walls, and cores. After the structural concept has been assigned the engineer estimates the approximate *capacity* and *serviceability* of its structural elements, e.g. bending resistance and deflection for a beam. The result of the first three phases are the architectural concept and the structural configuration, which are represented in drawings to lay out the main functions of the building with its structure. The architectural drawings include the spatial arrangement of the rooms, their occupancies, and the main building elements with their materials, whereas the structural drawings depict the load-bearing elements, sizes and materials for them, and their arrangement in space. The main element sizes are needed for the design of the service installation, which includes the placement of service ducts. On this basis, the architect and structural engineer carry out their detailed design. The structural design process continues with the structural analysis and design of the structural elements, which leads to precise element sizes and required arrangement of reinforcement. The end of the building design process is marked by the completion of the drawings for the formwork and the reinforcement.

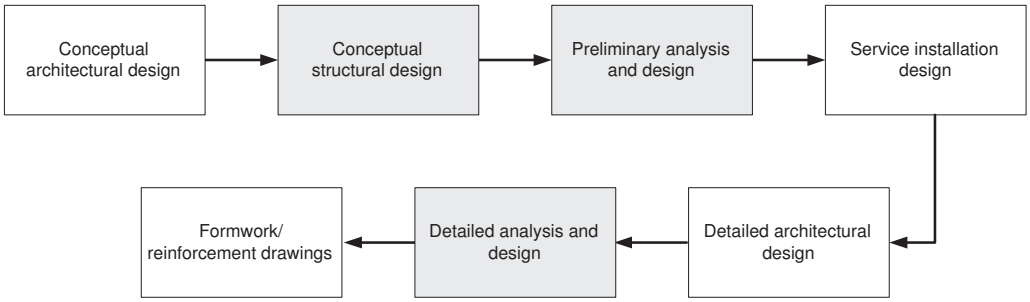


Figure 1.1: Phases of the building design process without iteration cycles

The *structural design process* is essentially a hypothesis formulation and test procedure because a structural configuration can only later be designed in detail if all information is available. If the engineer has deep *design knowledge* and starts with a correct structural configuration, reasonable estimates of the final element sizes and internal forces in the procedure, one detailed analysis followed by a design check is usually all that is required. HOLGATE calls the engineer's ability to design a structural configuration with a minimum of iteration cycles "the art in structural design" [Hol86] and FRASER "the conceptual process of logical thinking in which the engineering techniques are the base from which logic operates" [Fra81]. The design process consists of three interacting tasks: *conceptual design*, *preliminary analysis* and *design*,

and *detailed analysis* and *design*, see cf. [Fra81, Sch80]. Conceptual design has the goal to find a configuration of structural elements and systems that transfer the applied loads safely to the ground and stabilize the building. The structural configuration is usually called the *structural concept*[1] during the early design phase because details of the structural representation are omitted [Leo77], as the elements have to be designable later and comply with the geometrical constraints imposed by the architect [Sch80]. Preliminary analysis and design consists of simple calculations performed with an ordinary scientific calculator and based only upon the elementary concept of structural equilibrium and stiffness - the type of calculations that can be performed even "on the back of an envelop" - to obtain reasonable estimates of element sizes and main reinforcement. The detailed analysis/design constitutes much of the day-to-day activity in an engineering office is a later stage in the total design process. The detailed analysis of statically indeterminate structures requires a prior statement about the distribution of element stiffnesses, or the relative sizes of the element members of the structure calculated in preliminary design. A typical assumption is that bending rigidity is constant. The results from this analysis are then used to proportion the elements and choosing the detailed amount and arrangement of reinforcement in keeping with the initial assumptions about stiffnesses. Quite often the resulting element sizes violate these assumptions so a revised analysis, followed by modification to the element sizes, is required.

1.2 Industrial Motivation

Conceptual design is one of the most demanding tasks for the structural engineer during the whole design process [Wom02]. At this early stage, almost all important decisions about the structure are made in terms of robustness, costs and life-cycle functioning [BD95, Eng98]. However, a number of engineers, especially novices, lack design skills or disregard to design a structural concept because they do not maintain a *conceptual model* of the structure and/or rely on analysis and design tools for detailed design [dP01]. The impression of leading engineers, called "Prüfingenieure" in Germany, is that 1/3 of all practitioners design unsafe conceptual design solution due to a missing global load transfer and 1/3 of them disregard design constraints imposed by the architect or the DIN 1045-1 [Eis01b]. As a consequence, these shortcomings can result in the overall failure of the structure, unsatisfying conceptual

[1] Hubka [Hub82] calls a concept a sketched *interpretation* of a proposed structure. Therefore, I will denote the concept by $\mathcal{I}$.

design solutions, and hence, a highly inefficient iterative design process. This applies not only to complex design tasks but also to regular multi-storey designs of concrete building structures, which account for 80 per cent of all designs. Therefore, a simplified recommendation was adopted for prevailing design tasks in [dP03]. It implements the demand on engineers to design a *correct* structural configuration synthesized by *preconceived structural systems* from structural elements which must transfer the applied loadings safely to the ground, provide sufficient stability for the building, and satisfies the design constraints. The following example features some of the failures during early design stages that lead to the above mentioned shortcomings.

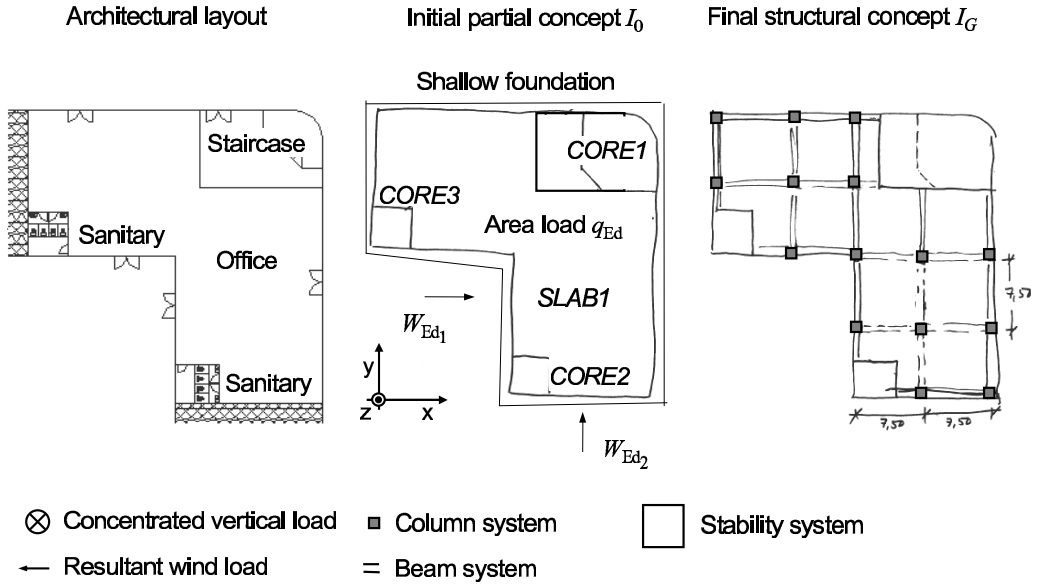


Figure 1.2: Elevation of a conceptual design task

Example 1 (Reasoning in conceptual design) Figure 1.2 shows a common conceptual design task, which is paper-based in current practice, adopted from EISENBLÄTTER [Eis01a]. Engineers usually extract first the important design information from a CAD-drawing. It depicts the room and occupancy concept. The drawing is the starting point of the design work. Experienced structural engineers build up an initial conceptual model of the structure, denoted $\mathcal{I}_0$. This results in the middle elevation sketch as external representation. The model includes building elements, which are supposed to be load-bearing, and external loadings by occupancy and other effects. Furthermore, economical and maybe material constraints are added to the initial working model because they determine the selection of element types. Becau-

se engineers often do not maintain a conceptual model, they miss to design a correct structural concept. They often do not establish load paths among the existing structural elements to find the critical members in the concept that dictate possible solutions. These critical members should be worked out first since all other design decisions depend on them. Since this is frequently skipped, the concept has to be redesigned later in the design process. This shortcoming could be avoided by maintaining a conceptual model about the structure to take the critical members and systems early in the design process into account. Experienced practitioners possess such a conceptual model about the structure. If they find that the structural concept is incomplete due to elements that are not appropriately supported or inconsistent due to elements that violate design constraints, they revise their design approach in order to reduce iterations. At the end of the design process, they have developed a conceptual goal model, denoted $\mathcal{I}_G$, that extends the initial incomplete model and depicted as the right elevation sketch, without iterating.

Designing a structural concept requires - demonstrated by example 1 - a model representing *conceptual knowledge* and the engineer's capability to reason about it. It is the first and critical step on which *control knowledge* can operate to reduce time consuming iterations by a suitable design approach in the process [Sim99]. Conceptual knowledge consists of two parts. The first part allows the engineer to assign the arrangement of elements and systems in the structural concept in terms of their required parts and properties stipulated in the DIN 1045-1. The second part helps him to assign valid load paths to the elements in the structure, similar to *qualitative reasoning* described in [Iwa97, For88]. Control knowledge consists of a suitable task decomposition of conceptual design into subtasks and the applicability of actions to develop a solution. The decomposition corresponds to the design approach (also called *design intent* or *design plan* [GGF94, Sim77]). Conceptual and control knowledge constitute the engineer's design knowledge.

According to surveys carried out by DEKKER and EISFELD & SCHERER [Dek00, ES02b], structural engineers miss support for conceptual design since commercial computer tools as *FLBuilding*, *SOFiCAD*, and *Nemetschek Allstar*, which are available on the German market, only support structural engineers at later stages in the design process. Research proposals for computer-aided conceptual design have not taken into account these requirements as they do not support engineers in the focused development of a correct structural concept (see for example the skeleton refinement method of HAUSER and SCHERER [HS97], and the decomposition method along knowledge hierarchies of FENVES *et al.* [FRG00]).

1.3 Definition of Research Problem

The aim of this research thesis is the development of a formalism, that can aid and guide the reasoning steps taken by engineers. This requires an extension of traditional support by a conceptual model and control knowledge for:

1. testing the correctness of a structural concept with its members during design,
2. sequencing subtasks according to the critical members before flexibly solving the subtasks in a bottom-up fashion, and
3. proposing actions to develop the concept and to reinstall the correctness of it.

I limited the envisaged design support to ordinary concrete building structures. They account for 80 per cent of todays buildings and feature the following characteristics: good soil properties, facade and stairwell design are non-governing load-bearing elements, no frame structures, living, shop, and office occupancy, simple layouts that can be composed of basic geometrical shapes, ordinary wind and snow loading, only walls, slabs, staircases and sanitary cores are given as load-bearing elements as design input, and layout variations are of a simple type.

1.4 Research Hypothesis

The computer-aided design system proposed in this thesis overcomes the aforementioned shortcomings in conceptual design for the domain of concrete building structures. The system is based on a model of conceptual design that takes the conceptual model of the structure and control knowledge into account. To realize the system for the domain specific requirements, a hybrid formalism that applies the description logic language $\mathcal{ALCQI}(\mathcal{D})^-$ with the reasoning services of *consistency* and *completeness testing* and the hierarchical task planning language SHOP that allows interactive *task decomposition* and suitable *action application* to implement a subtask have been chosen. The formalism:

- supports the work with incomplete concepts during design due to open world semantics[2],

[2]Open refers to absence of information that indicates lack of knowledge, whereas closed interprets the absence of information as negative information [BHS93].

- tests the consistency of a structural concept with its members on design constraints of the DIN 1045-1,
- identifies members with an incomplete load transfer due to closed world semantics at selective points, and
- sequences the conceptual design process on control knowledge to reduce iterations.

These results of the thesis were obtained by a work programme that determined the structure of the thesis.

1.5 Structure of Thesis

Chapter 2 starts with reviewing generally accepted definitions of conceptual structural design. They serve as starting point to discuss related research proposals with their open questions in detail. As the research about conceptual structural design is in its infancy, relevant classical design theory is briefly reviewed to define the problem of conceptual design of concrete building structures and to formulate a solution model for it.

Thereafter, survey data, that come from design studies in the domain of building structures [Eis00, Eis01a] is analyzed on the *knowledge-level* [New82]. It allows to view the engineer as a *dynamical control system* [New90], whose behavior is determined by its rationality trying to design a goal concept without iterating by its given knowledge. The analyzed design studies are employed twofold. Firstly they are used to refine the model formulation as knowledge-intensive search. Secondly, the practitioners' design knowledge was used to base the "designing as search process" upon it. Only a very few design studies where conducted. They were used to assign the most important "engineer's system" parameters that affect his performance. These parameters are a first approximation of the parameter space as the appropriate formulation of the model was the main focus of this thesis. For a critical discussion about when to choose few or large data sets for model formulation and validation see cf. [Sim96, Sim77]. The model is validated by means of a developed prototype in usability tests with engineering users. The results of the usability tests and the application scope of the developed model are discussed on a sample session in Chapter 4.

In Chapter 3, a formalism is developed on the basis of the defined model to aid engineers in conceptual design. Therefore, different relevant knowledge-based

methods for configuration design are reviewed. *Knowledge-based methods* are based on the idea to acquire expert knowledge and make it accessible to other users to support them. Thereby, they try to improve users' performance on the task at hand. The methods are discussed in terms of their applicability to conceptual structural design. After their evaluation, I select a logical and a planning method to implement the formalism. *Expressive description logics* appear to be an appropriate tool for conceptual design support because they allow the formal representation of and reasoning about incomplete structural concepts in form of a conceptual model in a correct way. *Hierarchical task planning* can be used to decompose conceptual design into subtasks beforehand and thereby decrease the number of iterations. The combination of these methods results in the interactive description logic based hierarchical planning formalism. To suit the domain specific requirements introduced by examples I first select two appropriate languages from the two groups of description logic and planning languages to represent the domain.

The resulting design language with the respective sublanguages for representing conceptual and control knowledge is formally introduced and the respective reasoning services for consistency testing and completeness testing of a structural concept as well as task decomposition of conceptual design and action application for a subtask are defined. Completeness testing is based on local closed-world semantics to check the load transfer of structural elements. Furthermore, interleaving task decomposition and execution of actions is supported to change the sequence of remaining actions for a subtask based on the developing concept. The services are used to provide the envisaged design support. To implement the services, the system employs the description logic system RACER and a plan-space search system SHOP. The description logic reasoner was developed by HAARSLEV and MÖLLER [HM01] and the planning algorithm by NAU *et al.* [NCLMA01].

To highlight the extended design support in Chapter 4, a prototype that draws on the developed formalism is presented by applying it to a conceptual design problem. To validate the model and system's scope of application the results of usability tests with structural engineers are discussed. Depending on the level of experience, practitioners report that they can detect incorrect and reduce time consuming iterations by the system prototype.

Chapter 5 summarizes the main ideas of my thesis and gives an outlook on open research questions.

Chapter 2

Model Formulation

Conceptual design of concrete building structures is introduced based on a literature study. Based on the result of this study and a detailed conceptual design example from practice, I specify the requirements for a model to support the engineer in this task and discuss which kind of support is not provided by previous work. The requirement specification helps to formulate a model that comprises a conceptual model of the structure, the subtasks and action types of conceptual design, and control knowledge to sequence the subtasks and actions appropriately. Analyzed design studies of experienced practitioners serve the model refinement by acquired conceptual and control knowledge. Because complete descriptions about action types and knowledge at early stages are rare in the domain, I briefly review accepted descriptions from design theory to base the model formulation upon it.

2.1 Conceptual Design

Conceptual design is an important and knowledge intensive problem solving task for engineers. However, it has not been formally defined in the structural engineering field. Only declarative descriptions in vague terms have been introduced that deal broadly with the characteristics a structure has to obey, cf. [Eng99, HB85, SE94, Fra66, Fra69, KW95], and the decomposition of conceptual design into subtasks that require different intellectual engineering activities, cf. [Leo77, BD95, Hol86, Fra81, Wom02, Sch80]. A missing notion of a structural concept with global load transfer and how to proceed while designing it results often in poor design solutions, which have to be revised by external experts [dP03]. The goal of this chapter is to define a *model* of conceptual design of concrete structures under the domain assumptions

stated in Section 1.3 because the computational support requires an appropriate and precise model formulation. DAVIS, who calls a model a problem, points out that "a problem formulation has an independent value, apart from the program, as an expression of one expert's mental model[1] of the task and the relevant knowledge" [Dav99]. A formulated model provides thus a basis for novices to understand the underlying expertise of experienced engineers [Mer02]. I start with descriptions of a structural concept that can be found in literature.

The reviewed literature describes the characteristics of a structure but not of a structural concept. This comes from the fact that traditional computational methods of engineering analysis and design are not applicable for defining a conceptual model of the structure that has to be specified on an abstract level with incomplete information. I summarize the descriptions by introducing the most important notions about a structure. For example, SCHODEK defines a structure strictly in terms of its load transfer requirement.

> "A structure can be conceived of as an organization of positioned constituent elements in space. No matter how some individual elements are located and attached to one another, if the resultant configuration and interrelation of all elements does not function as a whole unit channeling loads of all anticipated types to the ground, the configuration cannot to be said a structure" [Sch80].

O'BRIEN and DIXON describe a structure in similar terms but state the additional requirement of stability. They understand structural systems as groups composed of elements to perform a specific function.

> "A complete structure can incorporate any number of independent systems all of which act together to transfer the applied loads to the foundations and provide overall stability to the structure" [BD95].

HAMPE and BÜTTNER describe the load-bearing structure as a model of the real structure because the system is not known at the design stage [HB85]. The load-bearing behavior of the model describes the concrete processes taking place in structural elements, which are ordinarily given by a set of equations defined on concrete-valued state variables like support reactions and internal forces. Other textbooks emphasize the requirement on each structural system or element to carry, resist and transfer the applied loads to

[1] A mental model includes usually a conceptual model of the domain of discourse and knowledge about its manipulation - in the thesis domain knowledge about the design process.

adjacent structural elements, see for example [Eng99]. Hence, a structure is determined by the behavioral processes that take place in a structural system or element in response to a loading. These processes are usually casted into a system of equations in traditional equilibrium statics. They can be computed by structural analysis, since traditional engineering methods like stiffness, energy or equilibrium methods can be employed. However, these analysis methods cannot be used for conceptual design because they need precise information about the structure in terms of its topology, stiffness and loading.

In the beginning of the design process almost nothing is fixed and the engineer has to choose the type of structure, the principal arrangement of main structural elements and structural materials by experience and rules of thumb [Eng98, Leo77]. Therefore, the notion of structure of later design stages, which draw on detailed information, needs to be adapted to the conceptual design stage, since the *mereology* and the *topology* are the main outcomes of conceptual design but usually disregarded at later design stages. The mereology describes the decomposition of the structure into parts like structural systems and elements with their properties [AFGP96], and the topology specifies the arrangement of adjacent structural systems and elements interrelated via their supports [BA97]. Only the topology of single structural systems is mapped into incidence tables for structural analysis but not for the structural configuration as a whole unit. However, the arrangement of structural systems and elements by support relations is important for establishing global load paths through the structure.

The incomplete information demands an adaption of equilibrium equations and support conditions to conceptual design. These modified equations and conditions that constitute conceptual knowledge allow the engineer to configure a structure by arranging structural elements, which transfer the applied loads safely to the ground. I denote the resulting level at which the structure is represented *conceptual level* in comparison to the analysis level mentioned above. On this level, the equilibrium equations and support conditions find their counterpart in logical equations over the abstract domain of structural elements and systems. This abstract representation allows the engineer reasoning about how structural elements have to be chosen and arranged in structural systems with very little information. After the engineer has established a structural configuration, he estimates critical member sizes on approximately calculated internal forces at the end of conceptual design. A simple conceptual design example shall clarify the difference between the conceptual and later design stages.

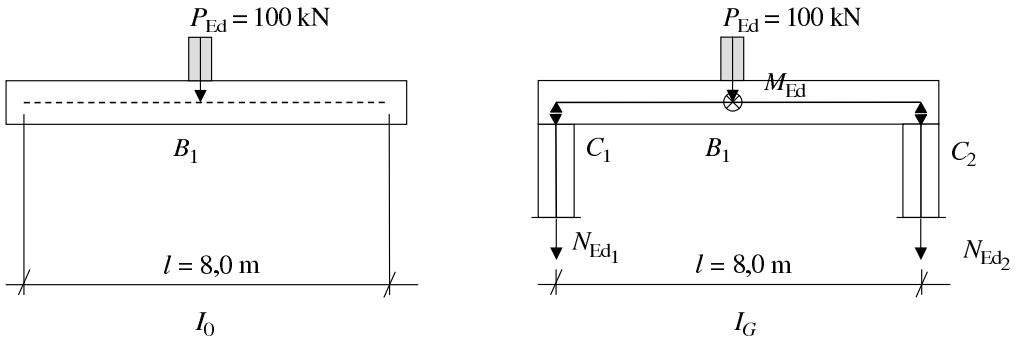


Figure 2.1: Incomplete and complete structural configuration

Example 2 (Conceptual reasoning about structure) In figure 2.1 an initially incomplete conceptual model $\mathcal{I}_0$ and a complete conceptual model $\mathcal{I}_G$ of a structure are depicted. It is the task of the engineer to complete the initial model by *parts* to a configuration that transfers the *load* $P_{\mathrm{Ed}} = 100\,\mathrm{kN}$ applied at mid-span and the *self-weight* through the beam *element* B_1 to the ground. For simplicity, it is assumed that the moment capacity as beam *parameter* is decisive for choosing its gross sectional constants. The self-weight is omitted for brevity. At the conceptual design stage, first *elements* have to be chosen that *support* the beam to ensure equilibrium, before the engineer can calculate internal forces and support reactions as *state variables*. In this example, he decides to simply support the beam by two columns C_1 and C_2. After he has added the two columns, he calculates the two support reactions that are equal due to symmetry and result in the actions N_{Ed1} and N_{Ed2} on the columns imposed by equilibrium statics, denoted $\Sigma V = 0$ with V the vertical forces:

$$\Sigma V = 0 : P_{\mathrm{Ed}} = 100\,\mathrm{kN} = N_{\mathrm{Ed1}} + N_{\mathrm{Ed2}} \text{ with } N_{\mathrm{Ed1}} = N_{\mathrm{Ed2}} = 100/2 = 50\,\mathrm{kN}.$$

Now, he is in the position to calculate the unknown value of the internal bending moment M_{Ed} by moment equilibrium, denoted $\Sigma M = 0$ with M the bending moments on the gross section at mid-span:

$$\Sigma M = 0 : M_{\mathrm{Ed}} = P_{\mathrm{Ed}} \cdot l/4 = 200\,\mathrm{kNm}.$$

Thereafter, the engineer estimates the beam depth z on the basis of the ultimate limit state *constraint* $M_{\mathrm{Ed}} \leq M_{\mathrm{Rd}}$ for providing the required moment capacity that resists the calculated internal bending moment M_{Ed}. He calculates the beam depth according to the DIN 1045-1 to:

$$z = \sqrt{\frac{M_{\mathrm{Ed}}}{\mu_{\mathrm{Ed,lim}} \cdot f_{\mathrm{cd}} \cdot b}} = 0,32\,\mathrm{m}.$$

As illustrated by the example, the engineer uses first conceptual knowledge to assign structural elements and relate them by supports with incomplete information. Thereafter, he calculates internal forces and support reactions by equilibrium statics, before he can estimate these gross sectional constants that are critical for member sizing in terms of stipulated design constraints in the DIN 1045-1 [Bau01]. In the following, I will describe conceptual design with its subtasks based on the structural engineering textbook sources.

In the literature, conceptual design is mostly seen as a synthesis process, which comprises different *subtasks* that draw on the experience of the individual engineer, cf. [Eng98, Wom02]. Subtasks usually decompose into subtasks at lower abstraction level until an atomic level is reached on which the engineer develops the structural concept. In contrast to the structure, the subtasks and actions involved in design are only sketchily described in the literature, cf. [BD95, Leo77], and, hence, provide a weak basis for deriving guidance how to design a robust structural concept from scratch. In addition, they leave out the notion of control knowledge for the suitable decomposition of conceptual design into subtasks at different abstraction levels to efficiently design a concept. I will give the most important descriptions here; the interested reader is referred to the references for more details.

FRASER understands conceptual design as an engineering process, that requires different solution techniques, which are highly interwoven. Conceptual design can be seen as comprising several subtasks dealing with different design issues.

> "In oversimplified terms conceptual design requires: the recognition of constraints and parameters relevant to the project, the assembly of constraints and the identification of their relationships, the devising of a structure to comply with the demands of the constraints" [Fra81].

This notion is shared by most authors in combination with the requirement of an appropriate selection of structural schemes that comply with these constraints, see for example O'BRIEN and DIXON [BD95]. WOMMELSDORF states that it is the most complicated task for the engineer, which is worked out in collaboration with the architect. The engineer has to decompose the building structure into single structural elements or systems that are designable according to design procedures [Wom02]. This opinion is also shared by LEONARDT, who emphasizes that conceptual design involves a lot of experience [Leo77]. SCHODEK places the emphasis of conceptual design on adapting the structure to the envisaged spaces, which are dictated by the programmatic function where structural elements have to be positioned and their interrelations be formulated [Sch80].

What kind of reasoning practitioners do based on the conceptual model, design constraints, and control knowledge and how each part of it contributes to the development of a correct concept without design iterations, is discussed in an example taken from a realistic project. It compares the design approach of an experienced practitioner to collected ones of engineers with little expertise. Along the approach taken by the experienced practitioner, I mention the occurring shortcomings[2] in the collected approaches. The example is an L-shaped office building situated in a city center between two neighboring buildings shown in figure 2.2, taken with permission from ARUP Consulting. For a detailed description of this project the reader is referred to [Eis01a].

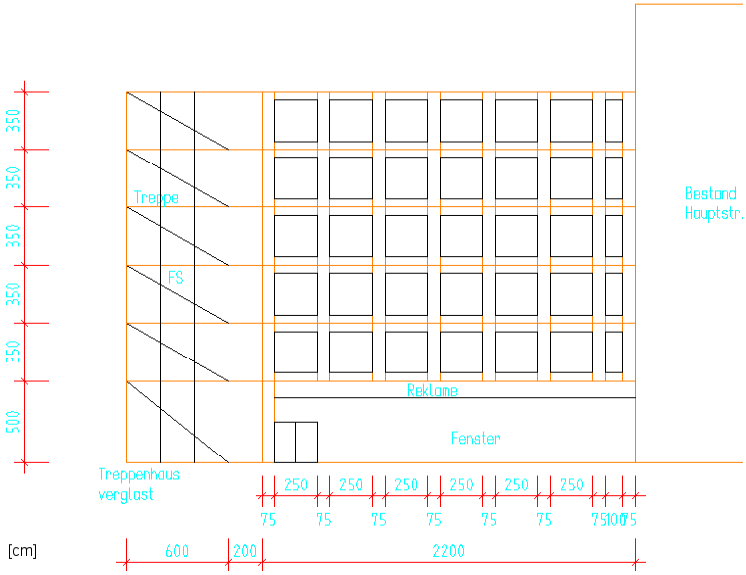


Figure 2.2: Side view of office building

The conceptual architectural design includes a number of drawing plans, which the architect usually hands over to the structural engineer together with the brief as design input for developing a structural concept. For brevity, only a plan of an upper office floor layout is depicted in figure 2.3. The plans are ordinarily received either in electronic or in paper form. They include geometrical building information and occupancies. Since conceptual design is paper-based in current practice due missing design tools, the engineer works directly with the architectural plans to design a structural concept.

[2]Shortcomings are marked by **R** because they set up the support requirements for a model of conceptual design.

Example 3 (Design project with shortcomings) The example comprises the task description for the engineer, the compared design approaches with the shortcomings, and the correct structural concept as solution to the task.

Task description: A structural concept for an office building with six storeys and one lower storey has to be designed. The initial room and occupancy concept is described by the following information:

- the floor layouts and side views that describe the building geometry,
- occupancies: shops at the surface floor, and offices at the other floors,
- good soil properties and no subsoil water, and
- design brief: flexible room concept, light facade, no load-bearing walls in staircase and shopping area, and medium budget.

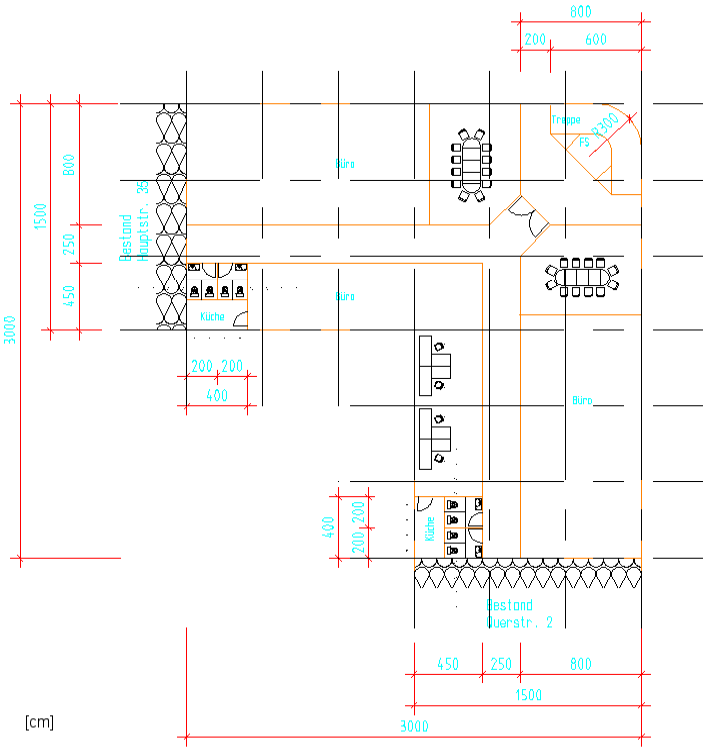


Figure 2.3: Office floor layout

The building has a base length of 30 m and a side length of 15 m, and is 23 m high. The ground floor has a storey height of 5 m and the five upper floors a storey height of 3.5 m, respectively. The offices can be reached via an aisle located in the middle of the building. The functional spaces are given by a grid of 5×5 m. Two sanitary shafts with 4×5 m are located at the end of the building. At the upper right corner, a staircase of 8×8 m with a light external glass facade is planned.

Design approach: The engineer starts to *develop the initial conceptual model* $\mathcal{I}_0$ as problem specification. He starts with the constraints and parameters. For the given storey height of 3.5 m a maximum slab system height of 50 cm is possible. The occupancy does not result in extraordinary loading conditions. Therefore, he selects for the shopping area a distributed load $q_1 = 5\,\mathrm{kN/m^2}$, for the office area a load of $q_2 = 3\,\mathrm{kN/m^2}$, for the roof coverage $g_1 = 2\,\mathrm{kN/m^2}$, and for the finish and services an additional slab self-weight of $g_2 = 1.5\,\mathrm{kN/m^2}$.

R: Some engineers miss to apply all slab loads that result from existing occupancies or finish and services like g_1. As result, they have to redesign all these elements and systems that participate in the load transfer for the modified slab loads.

The engineer chooses a column system for the vertical load transfer, which satisfies the flexible room constraint. Since no special constraints are imposed on the facade, a load-bearing facade is selected. He must design a local solution for the staircase because the glass facade around the stairwell does not allow for load-bearing elements.

R: Some engineers do not consider all existing constraints at the beginning. If an engineer does not consider the local solution for the facade at the staircase and places the load-bearing facade also there, he must calculate the slab system again because of wrong support conditions along the slab edges.

Since the soil is load-bearing the engineer plans to design an ordinary foundation with line and point footings.

R: Some engineers do not check the soil properties at the beginning of their design. If the soil has a low soil pressure, a base plate has to be chosen instead of line and single footings, and the whole structure has to be redesigned due to changing load paths.

In the next design step, the engineer *checks the feasibility*. He studies the floor layouts to find local conflicts that should be worked out as soon as possible because they could make a design of a robust structural concept infeasible. If this is the case, the engineer must contact the architect who

has to change the conceptual architectural design. Up to now, there exists only one local conflict at the surface floor because load-bearing walls cannot be placed internally to provide stability. Because he can place the shear wall on two columns at the surface floor, which do not have size restrictions, the conflict is resolved.

R: Some engineers skip to search for local conflicts like the one for the stability system or do it late in the design process. Both cases result in a redesign of the stability system that affects the analysis and design of the slab system above the surface floor.

The engineer starts to search for already existing load-bearing elements. In the corners of the building, the two sanitary cores can be used for lateral stability. By means of his conceptual knowledge about suitable stability systems he estimates if the existing two cores suffice for providing stability. Here, two additional shear walls have to be placed at the centre of the building to provide sufficient stability and to make the arrangement of the lateral elements symmetric.

R: If the two additional shear walls are not introduced the structure twists and the lability numbers might not be fulfilled. The shear walls act as additional support lines for all slab systems, which have to be calculated and designed again with the modified support conditions.

At this point in the design process, the engineer *plans the sequence of the remaining subtasks*. He decides to start with the vertical system and the possible spanning length of the slab. Thereafter, he is going to design the slab system and sizes the critical elements. Finally, he calculates the lability numbers for the stability system.

R: Because many engineers do not work out a problem specification they cannot plan the remaining subtasks. If the stability or the slab system has to be designed first depends on the existing lateral elements which provide supports for the slab system.

For the *design of the vertical system*, the engineer adds the existing load-bearing elements of the drawings to his developing structural concept. He starts with the external walls. The spanning length between the walls is 15 m. To obtain an economical slab height for a flat or a two-way spanning slab, which can span up to 9 m, he has to introduce another support line.

R: If a slab alternative, which spans outside its effective spanning length is chosen, the costs of the slab system increase. They make up 60 per cent of the total costs for the structure.

He introduces a slab beam as support line at mid-span and checks, if the beam can be placed at every floor. The floors are regular except of the staircase area where the column grid must be locally adapted. The column grid is thus 7.5×7.5 m.

R: Many engineers often do no check if they can provide the same load paths at all floors. If load paths vary between floors they have to redesign the vertical load-bearing system above the floor where the change in load paths occurred.

For the *design of the slab system*, a flat slab or two-way spanning slab has been already preselected. The engineer selects the two-way spanning slab beam alternative to span $l_i = 7.5$ m without problems. By the required slenderness ratio $l_i/d \leq 35$ for slabs stipulated in the DIN 1045-1 he calculates the height $h = 25$ cm which also includes concrete cover and half of the reinforcement diameter.

R: If the slab height is not correctly estimated by the slenderness ratio, the self-weight is incorrect. All design steps like calculating the lability number, sizing critical elements, etc. that depend on the self-weight of the slab have to be done a second time.

At the staircase, a local solution is required because the slab has a free edge along the boundary. The engineer designs a thicker slab edge. If this design is not sufficient he has to investigate other solutions like integrating the glass facade into the load-bearing system.

R: Some engineers do not consider or make wrong local adaptions due to changing geometries. However, the slab edge acts like a slab beam as support for the slab system. If the local solution is disregarded or incorrectly designed, the slab system has to be redesigned.

Load-bearing elements are *slabs*, *slab beams*, *columns*, and *walls* that the engineer *preliminarily designs*. For this subtask, a range of design tools and design procedures exists. For some preliminary designs, the engineer uses the computer for the calculations. He starts to determine the loads on the slab and its self-weight. The slab is only sized by the required slenderness ratio. Then, to obtain the required size of the slab beam, first the maximum internal bending moment over the support is calculated by $M_{\mathrm{Ed}} = -0.1 \cdot q_{\mathrm{Ed}} \cdot l^2$ from a design table where q_{Ed} is the slab load and the self-weight of the beam. Because a total slab system height of 50 cm is possible, the engineer calculates the required breadth $b = 100$ cm for the continuous slab beam with 3 spans on the fixed height of $h = 50$ cm (the breadth can be calculated in a similar way as the depth in example 2). For the columns, one internal column is designed at the surface floor because it is loaded by all the loads from the upper floors.

The loads of all upper floors are calculated based on the tributary areas to find the critical compression force. The engineer chooses a gross section of 50×50 cm for the column and calculates the required reinforcement. The external walls are not critical because a size constraint except of the minimum requirements for the wall breadth does not exist in the DIN 1045-1.

R: Sometimes, engineers do not calculate all critical members. If they miss to size the column with the highest compression force, they have difficulties to provide reinforcement in the detailed design that does not exceed the allowable reinforcement ratios of the DIN 1045-1.

The engineer checks if the actual reinforcement ratio does not exceed the allowable reinforcement ratio of 9 per cent for the column. For the ratio of 3 per cent, the column has some remaining load-bearing capacity which is favorable if any unexpected load increase occurs later at the detailed design stage.

R: Often allowable reinforcement ratios for elements stipulated in the code are exceeded. It results in the modification of the gross section at late design stages such that a second complete detailed analysis and design step becomes necessary.

As final design step, the engineer *checks* if the *stability* of the structural concept is in fact sufficient. The required gross sectional parameters of the shear walls and cores are calculated by the computer as well as the shear centre. The stability can be assumed as sufficient if the inequation

$$\alpha = \frac{1}{h_{\text{tot}}} \cdot \sqrt{\frac{E_{\text{cm}} \cdot I_{\text{c}}}{F_{\text{Ed}}}} \geq \begin{cases} 1/(0.2 + 0.1 \cdot n) & \text{for } n < 3 \\ 1/0.6 & \text{else} \end{cases}$$

for both building directions is satisfied, where h_{tot} is the building height, E_{cm} Young's modulus, I_{c} the total moment of inertia of all stabilizing elements, F_{Ed} the total vertical building load, and n the number of floors, respectively. For this building, the translational stiffness is sufficient an the rotational stiffness does not have to be computed since the stabilizing elements are placed symmetrically around the shear centre of the building.

Because there do not exist remaining conflicts and the load transfer has been assigned the structural concept is complete. The design of the staircase facade and the lower floor are not considered in more detail because they do not affect the structural concept.

Solution: On his *conceptual goal model* $\mathcal{I}_G$, the engineer builds up a structural concept that encompasses the following structural systems:

- the *vertical system* with an internal *column system*, a grid of 7.5×7.5 m, and external *load-bearing walls*,
- the two-way spanning reinforced concrete *slab system*, which fulfills the serviceability state limitations,
- the *lateral stability system* with two sanitary *cores* and the *walls* at the stairwell as parts,
- the *foundation* as *line* and *point footings*, and
- the critical *member sizes* of the *slab*, the *slab beam*, and a *column* at the surface floor that satisfy constraints of the DIN 1045-1.

Engineers often miss a conceptual model and control knowledge for designing efficiently a correct structural concept. They either design the concept non-globally that is load paths are discontinuous and incomplete or they do not elaborate the critical members or systems at the beginning of the design process. Both increases the number of iterations. Contrary to them, experienced practitioners first plan their design process on a *task level* before they develop a structural concept on the conceptual level. During design, they change locally their design approach for a subtask, if they find out that elements are inconsistent on the DIN 1045-1 or incomplete due missing supports. The initial conceptual model allows the practitioner to sequence the remaining subtasks to avoid iterations in the design process. Such iterations are redesigns of structural systems or elements that are affected by modifications of already designed members. Based on the feasibility check the remaining subtasks have to be performed in a suitable order because some subtasks need information that is created by others. The stability system has to be designed before all other systems if additional shear walls are required. If the slab system or the vertical load-bearing system has to be designed first, depends on critical elements governing the design of the systems. In addition, the slab height depends on the assignment of the column system that supports the slab system. The sizing of the critical elements makes only sense if the element types and their arrangement are fixed. For the calculation of the stability numbers, the total vertical building load is needed. Therefore this check can only be carried out at the end of the design process.

2.2 Requirement Specification

Motivated by the major shortcomings from current practice, the following list refines the requirements of Section 1.3 to model conceptual design for the envisaged computer support.

1. Testing the correctness of a structural concept:
   a) check constraints for structural elements and systems to ensure their consistency with the DIN 1045-1,
   b) support the assignment of load paths for each element to provide a complete vertical load transfer,
   c) assist the selection of suitable element types for given constraints.
2. Sequencing subtasks before solving them bottom-up:
   a) support the construction of an initial conceptual model that includes constraints like costs, soil properties, and loads,
   b) decompose conceptual design into a suitable subtask sequence based on the initial model.
3. Proposing actions to develop a concept and to reinstall correctness:
   a) reduce iteration cycles by selecting suitable actions to solve a subtask, and in case of incorrect member solutions, by proposing direct modifications to reinstall consistency,
   b) guide the user to members that are not complete in terms of their load transfer,
   c) force the user to solve conflicts directly after they have occurred.

Previous work will be discussed on these requirements in the next section.

2.3 Previous Work

Because a formal model with the necessary design input, the structural concept, and the involved task ingredients to develop a correct structural concept is missing a variety of computational paradigms to aid intellectual engineering activities has been applied in the structural domain, see [HL90, Mil01, MM94, RS03, SH97]. They understand design problem solving in the structural domain as constructing and working with some kind of

conceptual model on a symbolic or sub-symbolic[3] level, respectively [SH97]. MILES [Mil01] gives an almost up-to-date overview of applied methods to support conceptual structural design. The following list reviews relevant proposals in terms of the requirement specification and highlights the need for the knowledge-based design interface developed in this thesis.

- The automatic system HI-RISE [Mah84] was the first knowledge-based design system supporting structural engineers. Its purpose was to show that knowledge-based methods can be employed for supporting the structural engineer. The system and its successor systems discussed in [Mah91, MG96b] uses the PSRL reasoning system, which is rule-based and enhanced by a frame representation with function demons, for proposing preliminary design solutions of high-rise buildings. Later it was extended by a genetic algorithm to select optimal design parameters [MG96a]. The structural solutions are represented in the frame part, which included alternative refinements and decompositions according to the given control rules but no load transfer. The design task is subdivided into a fixed sequence of subtasks for structural systems. In turn these are separately realized by an automatic synthesis, analysis, evaluation, and selection cycle. This assumption that the design task can be decomposed into a predefined order of subtasks applies only to pure routine design problems like preliminary design. The interaction between the engineer and the system takes place at the selection level of completely specified solutions for which the structural elements and systems have been preliminarily sized by rules of thumb.

- The interactive system CONFIG [Gom98] developed in the SEED project [FRG00] assists engineers in conceptual design of simply-shaped building structures. Its main focus was to allow the rapid assembly of building elements that enclose a given spatial layout. It uses an object-oriented language for the representation and processing of the structural concepts. Conceptual knowledge, which was acquired from standard textbooks, is represented by decomposition and inheritance relations. The engineer can refine the overall building along a skeleton into basic structural subsystems called building entities but cannot test the overall concept for its correctness. Each building entity can then be designed by the application of so-called technology nodes that make up a graph for each building element. Methods for determining attributes, compositions and relations of the building entity under development

[3]Sub-symbolic refers to neuronal networks as for example trying to imitate the processing on the physical level.

are attached to the nodes in the search tree by so called technologies. Control knowledge is thus encoded in the refinement hierarchy. Technologies can be understood as actions. The applicability of a technology node is determined by predecessor nodes that must be already applied. The application sequence of technology nodes is fixed along the skeleton since nodes cannot change their location in the graph. Additionally, stored solutions can be retrieved for building entities, which the engineer has to adapt manually to the new designs.

- The system CADREM [KR97, Rap95] for conceptual design of structural layouts applies cased-based reasoning for retrieving a structural solution concept to the engineer. It was mainly designed for proposing structural grids in dependence of suitable slab types. The cases for slab types and the decomposition knowledge were extracted from textbooks. It represents cases for different subtasks by methods, which are applicable for certain preconditions. Thereby, the system generates solutions by methods used in solving similar cases in the past. The engineer can adapt the formerly generated solutions in a restricted way. Control knowledge is used for composing complete cases from partial solutions in the case-base. This framework is automatic in nature and the engineer can only adjust retrieved cases. Thus, a number of solutions can quickly be generated. It works for simple layouts when a conceptual model is not needed because similar cases can be stored as solutions in the case-base prior to their retrieval.

- The BGRID system [SMM99, SMM03] supports the engineer in generating alternative structural layouts for rectangular building structures with evenly distributed occupancy loads. Its main purpose is the rapid optimization of structural grids according to given design criteria. Its domain is restricted to steel framed buildings where heuristic knowledge exists in form of rules of thumb about span/depth ratios for seizing member dimensions. The knowledge and the underlying assumptions were elicited from structural engineers. It transfers the ideas developed by MOORE [Moo97] for conceptual bridge design to building design which requires the search for adequate alternative solutions. BGRID employs a genetic algorithm where span requirements and design criteria[4] are set up by the user prior to the search process. Thus, the support is automatic in nature as is the one of CADREM. Both systems work only for regular building layouts and simple loading conditions for which control knowledge can directly be used to compute structu-

[4]Criteria are constraints that are ordinarily fixed prior problem solving.

ral solutions. Its applicability is hence restricted to design situations with simple geometry and loading conditions. The main results of a BGRID application session include structural grid dimensions, a slab system with its assembly and dimensions, and the resulting vertical storey dimensions.

- HAUSER's system PRED employs a skeleton refinement method for assisting the engineer in the conceptual structural design task [Hau98]. It was designed for developing the required structural systems in a top-down process for rectangular building layouts. The structural solution model is dynamically constructed along a decision tree, which represents the conceptual knowledge about solution alternatives. The decision tree can be explored along the is-a and has-part relations of the envisaged concept. For the representation of the conceptual model a frame language is used and for the control knowledge an operator scheme from Artificial Intelligence[5] planning [HS97]. However, the sequence of possible actions is fixed due to the strictly confined operator applicability encoded by the operators' preconditions. A pure top-down assisted design process is to inflexible because engineers first plan their design, which is later bottom-up developed by direct element manipulation [ES02b]. Furthermore, the system does not take the load transfer and the design constraints of the DIN 1045-1 into account. For a detailed description of the limitations of the PRED system see [ES02b].

The thesis extends the conceptual design support according to the requirement specification because other proposals support only the engineer in 1.c) and 2.a). The newly developed computer-aided conceptual design system in this thesis covers support for all other points except of 3.c) and partly 1.c) because complex geometrical and costs constraints are excluded. The difference to the other proposals will be discussed in more detail at the end of the thesis in Chapter 5 after the prototype system has been presented. The kind of developed support in this thesis is termed *expert-critiquing*, as introduced by HÄGGLUND [Häg93]. The bottleneck for the success of such systems is usually to acquire and represent realistic expert knowledge about the task and the experts problem solving methods [GRS00], since most experts can only explicate implicit knowledge in a restricted way. Section 2.5 will discuss the problem of knowledge acquisition in more detail but first general design theory will be reviewed for the model formulation.

[5]Artificial Intelligence refers to the science and engineering of making intelligent machines, especially intelligent computer programs [McC00].

2.4 Existing Models of Design Theory

Design theory is concerned with developing a formal theory of design, which is domain-independent [Hub82]. This formal theory aims at broadening the capabilities of computers to aid designing, drawing upon the tools of artificial intelligence, cognitive science and operation research [Sim99]. In the past, the development of different models has not resulted in a shared formal theory. According to GERO this is because no present model does possess explanatory capabilities for describing all constituents involved in design [Ger98b]. However, all design models assume that a general human problem solving behavior is common to all engineers regardless of the design domain, where designing can be in principal seen as a form of *rational human problem solving* [NS72, GP89]. These assumptions are indicated by NEWELL's rational agent view [New90], which states the hypotheses that a human problem solver like an engineer can be described on the *knowledge-level* by its goal, actions and knowledge where the goal is to provide desired function. The principle of the agent's rationality now states that the agent applies all its knowledge to reach the goal, where knowledge can be understood as an external agent characteristic. GOEL remarks in [Goe94] that in a design process not all objects are known at the outset in comparison to ordinary human problem solving. He understands designing therefore as a more explorative process where engineers create new symbolic objects during problem solving not known to them beforehand. This limitation is however debatable since SIMON argues that the manipulation objects and the desired function must be at least intensionally as design knowledge defined, since otherwise engineers would not know whether or not they found a valid solution [Sim77].

In general, *designing* refers to the range of actions participating in the model to transform *function* into a *structural description* in such a way that the described structure can produce this function[6] [Ger90]. They are also often termed *activities*, cf. [Hac00, CRR+90, Suh90]. I will use the term actions in the sequel to denote external and internal activities, which comprise also planning actions to control the design process. The actions draw on knowledge to carry out the transformations. One of the most accepted definitions is by HUBKA, who describes in [Hub82] the task of designing as a search process, which "consists of thinking ahead and describing a structure, which appears as carrier of the desired functions under the given constraints". Further, he interprets conceptual designing as assigning possible structures that could fulfill the desired functions and constraints. A structural concept is therefore a hypothetical solution that might be later adapted at the detailed analysis

[6] Other authors call Gero's function "functional requirement".

and design stage. He calls a sketched interpretation of a proposed solution a *concept*, also termed *scheme* by other authors, see for example FRENCH [Fre99]. GERO realizes similar characteristics for conceptual designing.

> "An important characteristic of conceptual designing that is missing in detailed designing is that in conceptual designing not all information that is needed to be known to complete a design is known at the outset" [Ger98a].

Typical of conceptual design is that the engineer cannot verify the functions due to incomplete information but ensures that the global functions can be embodied by appropriate subsystems, where conceptual designing is also termed *functional design* [PB96, UT97].

Most of these models interpret designing, also termed *design problem solving*, as an intellectual human activity, which involves *search* and *exploration* of design state spaces along paths of design states. A *state* is a snapshot of the world at a certain time and serves as conceptualization a fixed purpose [NG89]. In structural design, a state is a concept description that might be incomplete and comprises a set of objects and relations among them. Objects are structural elements, and systems, e.g. stability system, supports and loads. At the conceptual design stage however, the engineer has only incomplete information about support objects and how elements/systems are connected via supports. Therefore, he represents a support object and a connection of elements or systems by a support relation that is later refined into a separate support object and connectivity relation. A set of support relations connect thus elements, loads and systems at fixed points to transfer the loads from their application points through the elements or functional systems to the ground. In addition, compositional relations aggregate structural elements to functional systems. In the literature, objects in a *design state* are usually described by a set of value assignments for *structural* or *behavioral variables* in which the functions manifest. In the literature, different notions of function are given, cf. [CJ00, UT97, Ger90, PB96]. I understand function as a *mode of deployment* in agreement with CHANDRASEKARAN. Here the function of an object is fulfilled, if the domain constraints among external and internal behavioral variables, represented by a set of equations, are satisfied for a given purpose [CJ00]. Domain constraints constitute thus the knowledge to evaluate the behavior of the solution in terms of the desired function. For a structural concept, the purpose is to ensure a global transfer of the applied loads to the ground by support relations, overall stability, and the local transfer by structural elements. The same applies to elements or systems for which their terminals can be considered as external variables of

the object to communicate to other objects. Supports take on this role for members in a structure.

Example 4 (Objects in structural design) In a design state of example 2, the beam object B_1 of an intermediate concept, denoted $\mathcal{I}$, is described by associating the bending resistance M_{Rd} as structural variable, and the internal bending moment M_{Ed} as behavioral variable to it. The element function can be understood as a Boolean function $\mathit{designable} : E \rightarrow \{\top, \bot\}$ with E the set of elements. It evaluates to true if the function of the beam to resist the internal bending moment, described by the relation $v_2 \leq v_1$, is satisfied by a value assignment. For the beam, $\mathit{designable} : B_1 \rightarrow \top$ since the value assignment $v_1 = 210$ kNm and $v_2 = 200$ kNm satisfies the inequation for the mode of deployment of the beam to resist the load. Because the elements are only approximately sized for the main functions in conceptual design, which have to be later designed in detail, the Boolean function is called designable. Structural and behavioral variables are called *parameters* and *state variables*, e.g. bending resistance and internal forces, in structural design. Furthermore, the beam B_1 is related to the load P_{Ed} by the relation $\mathit{supports}(P_{\mathrm{Ed}}, B_1)$ and associated to the vertical system S_V by the compositional relation $\mathit{hasPart}(S_V, B_1)$. Other constraints might arise from the building geometry or the requirements of the DIN 1045-1.

The *design state space* is then conceivable as the set of possible states that could exist if all the design actions, except of planning actions, operate syntactically correct on all the object variables to assign values to the variables and arrange objects by possible structural relations. Search takes place in a finite state space of enumerable solutions that is fixed prior problem solving whereas exploration accounts for a countable state space that cannot be completely enumerated before or even during problem solving [RN95, Pup91]. This notion usually results in the separation of the design problem class into *routine* and *creative*, also termed *configuration* and *conceptual design* [BC89, Ger90]. However, the boundaries are fluent and not fixed. In the building domain, the engineer composes a structural concept by given structural elements, and the design can, thus, be classified as explorative since the solution alternatives are constructed during design. It is also of routine nature since the structural concept is decomposed into a fixed set of structural systems, which are composed from a well-known set of elements described by variables.

In the following, I introduce selected models of designing. They are GERO's function-behavior-structure (F-B-S) model [Ger90] also proposed by UMEDA [UTY90], and SIMON's planning model [Sim99] extended by the task structure idea of CHANDRASEKARAN [Cha90] being widely accepted in the field

of design theory. Their aim is to draw on their characterization of designing, especially design action types to develop and test a concept, and planning actions to decompose and sequence the design task, to model conceptual structural design.

2.4.1 F-B-S model

The F-B-S model first proposed in [Ger90] and successively developed in [Ger98a, GK02] represents the developing design in different states. The model separates the information of a design state into *function* F , *structure* S , *actual behavior* B_s , *expected behavior* B_e , and a *description* D of the structure. Conceptual knowledge corresponds to expected behavior in this model to synthesize a structure and test the actual behavior. The different design information is linked by types of *design actions*, denoted A_D not including planning actions. By an action, information is transformed or deduced from known one by means of knowledge. The types of actions occurring in conceptual structural design can be described as depicted in figure 2.4.

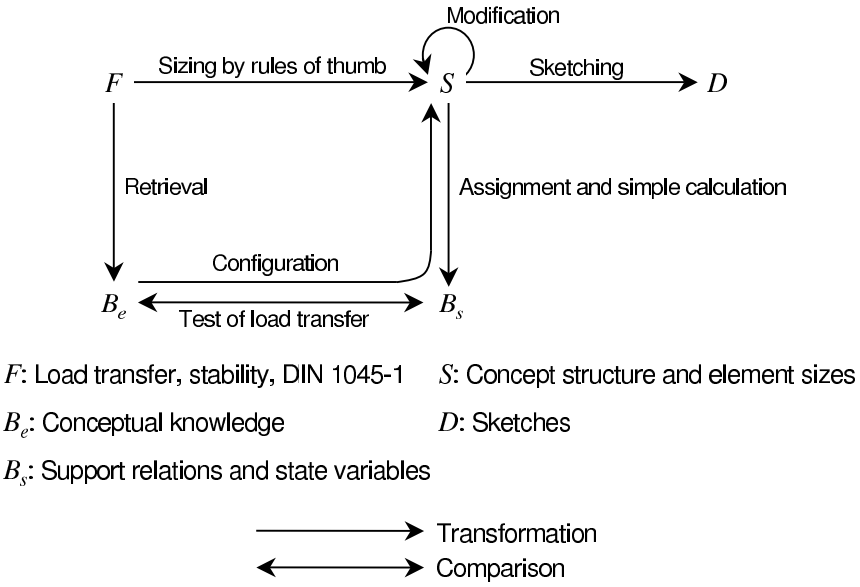


Figure 2.4: Actions types in conceptual design according to F-B-S model

Example 5 (Action types in conceptual design of structures) Consider the small design task from example 2. After the engineer had built up his initial model $\mathcal{I}_0$ of the structure, the engineer retrieved conceptual knowledge B_e for the function F to transfer the applied load P_{Ed} to the ground. In the next design step, he refined the initial configuration, here called S, by choosing the structural elements C_1 and C_2 and relating them with the beam B_1 by support relations. Support relations serve not only to configure the structure but also to ensure load paths among elements. Thereafter, he calculated the actual behavior B_s given by support reactions N_{Ed1} and N_{Ed2} of the columns, which implement the support relations of the conceptual level, and the internal bending moment M_{Ed} of the beam. He sized by a rule of thumb from the DIN 1045-1 the beam depth to resist the bending moment. In the testing step, he compared the expected load transfer B_e to the actual load transfer B_s. In the example, the expected and the actual load paths are the same and no modification step is necessary. If this was not the case, the engineer would have to modify the configuration. At the end, he produced a sketch D of the concept's structure on the basis of the conceptual model $\mathcal{I}_G$.

The final product of conceptual structural design is created by the above action types. It is a sketch D of the concept's structure S entailing the fixed functions F on the conceptual model. GERO emphasizes that a causal mechanism of the underlying structure is needed because direct mapping from function to structure does not exist except. The expected behavior as such is a causal mechanism on which the design of the structure is based. The causal mechanism shall explain an aspect of the behavior in terms of others in such a way that the aspect being explained can be changed if desired [IS94]. An explanation refers to parts of the structure and describes how the structure works in terms of causal interactions among the parts that lead to the actual behavior, termed *desired result* in [KB86]. Thus, a direction of inference is needed for reasoning in ordinary non-directional equations [Sim77]. The conceptual knowledge is necessary to evaluate the found solution and to make modifications to incorrect structural concepts during design.

Because the F-B-S model could not explain the design approaches of architects from analyzed design studies for conceptual stages [GT01], GERO and KANNENGIESSER [GK02] proposed an extension to accommodate developments of cognitive science into the F-B-S model. This resulted in the model of *situated designing*, which extends the model by two qualitatively new processes. These incorporate the notion of *situatedness* and *constructive memory*. Situatedness describes that actions are not the outcome of previously set up plans but result from the interaction with the situation the designer is in [Suc87]. Constructive memory subsumes the idea that memories are not fi-

xed by experience but are always newly constructed in response to the need of this memory [Cla97]. In the extended F-B-S model, situatedness is understood as the designer's interaction with the situation, which is represented by the sketches D. The interpreted sketches refer to the engineer's conceptual model of the structure in the current situation. The model serves also for the proposition of design actions for a focused subtask because the engineer conceives a subsequent situation resulting from possible design actions on his conceptual model. By this extension, designing also includes acting in reflection to the current design situation made up by sketches, which SCHÖN describes as "shaping the design in reflection to the situation" [Sch83]. Thereby, the sketches serve as an additional information source in a similar way as the long-term memory does. The sketches drive the design process to preferred solutions by recognition. I found similar results in the protocols of the structural design studies, which indicated that the engineer follows only an abstract design plan in form of a suitable subtask sequence. The local design approach in a subtask is however revised in response to intermediate sketched design solution. An intermediate design solution is a structural concept that is not completely designed.

I use GERO's categorization for the design action types to model design actions in the structural domain. I preclude the actions that formulate the conceptual knowledge since the experienced engineer simply retrieves it for the task at hand. This knowledge will be later stored as persistent knowledge in the knowledge base.

2.4.2 Planning Model

SIMON understands design as a problem of finding an action path from an initial design situation into goal one, since no direct transformation from the initial to the goal situation exists by a trivially constructible transformation matrix [Sim77]. A situation simply corresponds to a sketch of the design state because the world is static. SIMON hence states that "everybody designs, who devises courses of action aimed at to change existing situations into preferred ones" [Sim99]. Due to the limited capacity of his working memory the engineer cannot generate the entire design state space of possible solutions to find the optimal one. Therefore, he searches for promising solutions along an abstract design plan where suitable subgoals and design actions for a certain design state constitute his control knowledge about the design process. He calls these solutions "sacrificing" and the engineer's planned behavior *bounded rational* for the agent's utility to minimize the length of a design action sequence.

Example 6 (Planning in conceptual design) Consider the conceptual design task of example 3. The engineer creates first the initial conceptual model $\mathcal{I}_0$ to plan the design process. He chooses the following abstract plan represented by a list of tasks: vertical system design, slab system design, sizing of the members, and calculation of the lability numbers. Now, he starts with decomposing tasks into subtasks with greater detail until he has reached the conceptual level on which he develops the functional systems by design actions A_D of the F-B-S model. The abstract plan provides hence focus and reduces iterations in the design process.

On this basis, designing can be understood as a sequential, deliberative activity [New90, Joh02], which encompasses interleaved *problem formulation*, *problem finding* and *problem solving* [Sim95]. Problem formulation is the construction of an initial conceptual model for the task at hand, problem finding the setting up of the design goals, and problem solving the changing of the current state into a goal state, respectively. The state space corresponds to the solutions enumerable on the engineer's conceptual model. Goals correspond to subtasks, which have to be performed to reach certain design states. The design problem is continually reformulated during the process of designing in which goals by planning actions, denoted A_P, and constraints retrieved from long term memory emerge. Constraints are ordinarily expressed as the set of equations a solution has to satisfy. The constraints are equivalent to the conceptual knowledge at early design stages. The engineer generates alternatives by composing known components into an overall solution (in the structural domain elements into functional systems). Creative solutions can thereby emerge through combinatorics. The selection of components takes place in the course of design by recognition. The environment given by sketches provides the stimulus for retrieving structural element and system alternatives from long term memory or external information sources. The sketches aid the engineer as external memory and accumulate design information about the current design state. Thus, the engineer records decisions made in the sketch and can review them in different design states. Visual cues thereby evoke relevant information. The design process follows a Choose-Record-Review-Recognize-Revise cycle (it is another view on the design action types as introduced by GERO) from the highest abstraction level to more detailed levels. By this kind of recognition driven design process the engineer takes all considerations into account, step by step, during designing. The incomplete conceptual model allows to retain flexibility because the engineer cannot anticipate all contingencies.

Planning is thus represented in SIMON's model by subgoaling (setting up subtasks) and by selection of applicable actions to solve a subtask. Since

this model misses a clear notion about how to plan a hierarchical design by task decomposition, I extend it by the task structure approach from CHANDRASEKARAN. It is based on the idea that a design task with its task structure can be roughly modeled as a decomposition tree [Cha90, CJS92, BC89], first proposed by MOSTOW [Mos85]. Nodes in the tree refer to *simple* or *compound tasks*, denoted t. By a planning action on the conceptual model, a compound task decomposes, e.g. conceptual design, into a sequence of subtasks on lower abstraction level $t \rightarrow [t_1, \ldots, t_n]$, whereas simple tasks, e.g. design slab system, are implemented by suitable design action sequence from the F-B-S model $t \rightarrow [a_{D_1}, \ldots, a_{D_n}]$, which are ordered according to the *applicability* relation over design states. A design action is applicable in design state if its precondition is entailed by the state. Control knowledge is thus represented by suitable task decompositions of conceptual design into task sequences [CJ93] ranging from the concept to functional systems to elements and the design actions preconditions. A task decomposition for a compound subtask in conceptual structural design is similar to a *design episode* introduced by HACKER [Hac00], which represents a local design approach to solve a certain subtask. It allows the engineer to organize the design process in a flexible way on the task level.

I employ the idea of hierarchical task decomposition and action application to model the engineer's design approach. It reduces iterations at the task and conceptual level.

2.5 Design Studies

After the action types for design problem solving in the structural domain have been selected knowledge acquisition is the critical step to formulate a suitable model[7] of conceptual design. Therefore, design studies and interviews are used to acquire the conceptual and control knowledge, the subtasks at all abstraction levels, and the instantiations of the action types. The explicit goals of the complete knowledge acquisition process are stated below.

- Elicitation of the precise information needed for the initial concept $\mathcal{I}_0$, when the engineer starts the conceptual design task and plans subtasks.
- Refined definition of a correct structural concept $\mathcal{I}_G$ as output of the design process.

[7] Since the model provides the foundation for the formalism that will become the core of the system, the system's model should closely match the engineer's mental model [FH90].

- Enumeration of a set of external actions (sketching and calculating) to develop structural systems and elements in subtasks, and a set of internal actions (reasoning actions) to test the consistency/completeness of a concept and to sequence the subtasks of conceptual design.
- Conceptual model with knowledge about structure, global load transfer, structural object parameters, and external loading conditions to construct a concept and test if a concept is consistent and complete.
- Control knowledge about the decomposition of tasks into subtasks and the applicability of actions to subtasks and intermediate design solutions.

The knowledge and the actions are stored in the knowledge base as persistent knowledge, which the engineer does not modify during the course of design due to the preclusion of learning. Creating the external representation in form of sketches is included in the actions for configuration and modification by structure. In the following, I describe the assumptions made for setting up and analyzing the design studies.

2.5.1 Assumptions for Knowledge Elicitation

I assumed for the knowledge acquisition process that the engineer can be modeled as a *rational agent* [New82] and the observer and the practitioner share a common symbolic representation of the structural concept. This assumption was also indicated for the task environment of design by SACHSE and HACKER [SH95] who classified design problem solving as *knowledge-based acting*. There it was found that engineers act internally and externally during the course of designing, because they apply all relevant and accessible design knowledge for reaching their design goal, in the domain of the thesis a correct structural concept and a design process with as few as possible iterations. However, due to the limitation of the engineer's mental processing capacity planning and opportunistic design approaches are combined on the basis of the current state [Hac00, Dör01].

These assumptions are of special importance for the model formulation and the knowledge acquisition from design studies. First, the model formulation based on the analyzed design studies is independent of the underlying implementation of the design task on the computer. Thus, the representation of the task with its knowledge constituents - preferably by logic - and its symbolic processing on lower levels can be clearly separated [New82]. Second, the

design knowledge can thereby be inferred from the a priori known engineer's goal of efficiently designing a correct structural concept, since the observable problem solving behavior in form of design actions is determined by the engineer's knowledge.

I neglected perceptual and motor skills and presumed a sequential goal subgoal problem solving behavior, because I focused on eliciting the deliberative part of the design activity [Joh02, NS72]. I also excluded any type of learning. Furthermore, I considered the engineer as a feed-back control system defined by CHANDRASEKARAN for human problem solvers in general [Cha03]. Thereby the engineer's behavior is determined by the system parameter of design knowledge. By studying practitioners with a high level of design knowledge, the behavior is more stable and allows more simply to elicit an underlying system structure, in the thesis the engineer's mental model of conceptual structural design. The engineer as a control system plans actions by information from the current design state. Information retrieved from his long-term memory or from sketches continuously control his design approach to achieve the design goal. Architectural constraints and requirements of the DIN 1045-1 are only used for selecting an alternative from the range of possible ones. The engineer can hence be described on the knowledge level during design problem solving by: a symbolic representation of incomplete structural concepts that are observable in sketches, the desired goal state including a correct goal concept and a short design action sequence, the conceptual and control knowledge, and the design actions [ES01a]. Reasoning about the consistency/completeness of the concept and planning the design process by task decomposition and suitable action selection on top of it arises thus from the need of controlling his behavior to reach a goal state. The reasoning actions on the symbolic representation will be realized by appropriate inference services on the computational level.

2.5.2 Elicited Knowledge

Since different knowledge acquisition techniques are suitable for certain knowledge types [Bur98, Fri93] we carried out separate sessions for knowledge about the structural concept [ES02b] and knowledge about the design process [Eis01a, Eis00]. Conceptual knowledge independently defines the set of conceivable structural concepts. They can successively be developed by external design actions and tested for correctness by internal actions during designing. It thus lays the foundation the engineer acts upon. Control knowledge in form of suitable task decompositions and F-B-S actions in a state can be afterwards described on top of the conceptual model to reduce design

iterations by planning. I conducted a range of document analyses, design studies and interviews with engineers, where protocol techniques were employed which can be found in [CCD96]. Before conducting the design studies, two running design projects were observed in order to use real world conceptual design situations in which structural engineers are embedded. They served choosing appropriate working environments for the design studies by protocol analysis. SCHÖN argues that designing is mostly transacting with design objects and relations among them within design worlds [Sch93]. Therefore, I tried to put engineers into their natural environments for the design studies in which they could make up their own design worlds by sketches and hand calculations as memory aids, depicted in figure 2.5.

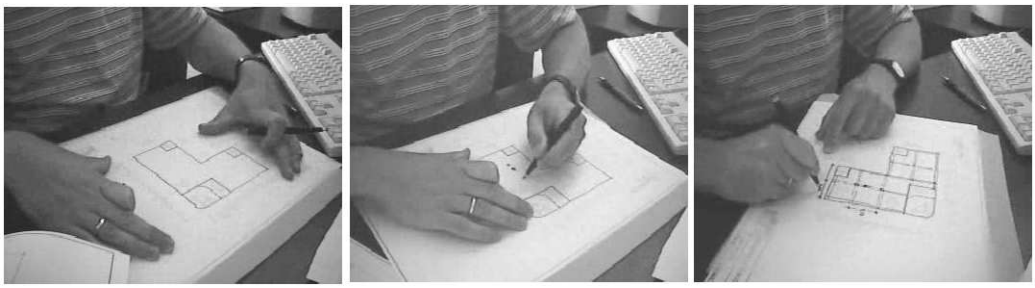


Figure 2.5: Environment during conceptual design

Since I focused on finding a common structured design approach, I worked with practitioners that could draw on deep domain knowledge for developing a structural concept and planning the subtasks. They feature a stable design problem solving behavior in comparison to novices, which tend to design in a less structured way resulting in a varying design problem solving behavior over a set of similar samples [KG02]. Hence, my work is different from the one about analyzing the difference of conceptual design problem behavior among novices and practitioners carried out in [KG02], because my goal was to find a common rigorous design approach shared among experienced engineers. It requires from the practitioner to possess a clear mental model of the structural concept and the design process.

First, the conceptual knowledge acquisition resulted in the establishment of a robust domain representation for structural building concepts that was capable of covering real world cases from practice under the restrictions stated in Chapter 1. The knowledge corresponds to expected behavior B_e of the F-B-S model and is represented in a description logic. It encompasses a domain taxonomy and a description of complete load paths for elements.

The taxonomy included for testing the consistency of a structural solution:

- the set of basic structural elements described by parameters, state variables, and simple constraints for selection of element types,
- the set of external loads, described by allowable concrete-valued intervals for ordinary building structures,
- the minimum and maximum gross section dimensions and reinforcement ratios for each structural element of the DIN 1045-1, and
- the admissible decomposition of structural systems into elements.

To test the completeness, the description of the load transfer comprised:

- the admissible arrangements of elements by support relations, and
- the main ultimate limit state requirements for structural elements and the stability system stipulated in the DIN 1045-1.

Second, four project documentations of already finished projects were analyzed. On their basis, the initial concept, which included external information given by the architect and other standard information gathered by the engineer, was elevated. In addition, the analyzed design documents determined the information that a goal concept has to comprise. In the following, the required information about the room and occupancy concept, and the initial-, and the goal concept is informally summarized. The room and occupancy concept included: architectural plans (elevations and sections), facade type, internal space concept, service installation requirements, constraints like costs limits or geometrical restrictions for floor heights, and occupancies. The initial concept was augmented by design information that engineers simply look up in codes or drawings resulting in the *initial concept* $\mathcal{I}_0$. It was comprised of: occupancy loads, self-weight, and external loadings according to DIN 1055, assigned load-bearing conditions for external walls, the foundation type, local conflicts and governing constraints. The *goal concept* $\mathcal{I}_G$ as a consistent and complete structural concept documented in general the following: vertical system with all its subsystems, slab system, lateral stability system with cores and shear walls, decisive structural elements with gross sectional constants, roof type, and the support and compositional relations. I elicited information about the relations by additional interviews with practitioners. Furthermore, rules of thumb for calculating parameters as the gross sectional constants and state variables were also acquired from the written project documentation.

For the appropriate definition of the conceptual design task with its actions, subtasks, and control knowledge, and how they are interrelated a second run was conducted at an engineering office by EISENBLÄTTER [Eis01a]. Two design studies with protocol analysis methods were conducted. Both studies were supplemented with many interviews because not all knowledge could be elicited directly from the analyzed design studies. Figure 2.6, taken from [Eis01a] shows the distribution of the design actions along the time in regard of the design progress.

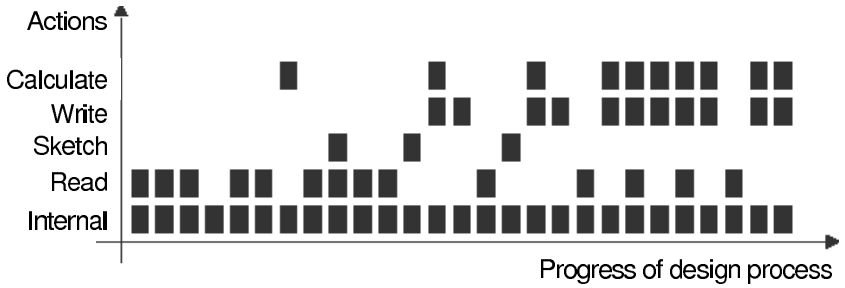


Figure 2.6: Mean response over two design studies

It can be seen from the diagram that in the beginning internal actions dominate, whereas at the end of the design process external actions prevail. External actions are employed for constructing the structural concept observable in sketches and for calculating. Internal actions are usually used for testing the consistency/completeness of the concept under development and for planning the subtasks according to the partial concept. However, whether or not an action is external or internal depends on the expertise of the engineer. Actions for information gathering like reading the design documents and communicating with the architect cannot be supported and are therefore excluded from the listed actions. One design study employed the thinking-aloud method to capture the actions. They are given in the following.

The *configuration actions* $B_e \rightarrow S$ are: select overall building information, make building storeys with occupancies, apply external load, create a new structural element with location from scratch, copy an existing slab or vertical load-bearing system and change a structural aspect, decompose slab or lateral stability system into elements, synthesize a vertical load-bearing system from elements, and establish support relations among elements.

The *sizing actions* by rules of thumb $F \rightarrow S$ are: set or determine a system's or element's attribute value, calculate a system or element parameter according to a design procedure from DIN 1045-1.

The *assignment and simple calculation actions* $S \rightarrow B_s$ are: calculate internal forces or support reactions by domain equations.

The *modification actions* $S \rightarrow S$ are: delete or modify a load, a storey, a system, an element or the building information, respectively, and change support or composition relations.

The *test actions* $A_T \subseteq A_D$ for load transfer are $B_e \times B_s \rightarrow \{\top, \bot\}$: test at the end of a simple subtask if the system transfers the applied loads, and check the consistency of an element or system during design according to the DIN 1045-1 requirements including the calculation of lability numbers.

The *planning actions* are: sequence the subtasks on the basis of the initial concept and select design actions on the intermediate structural concept.

The set of all actions is denoted A. They will be modeled in the next section, and, in Chapter 3, translated into suitable symbolic actions for the formalism. A second study used a retrospective method for eliciting the design process organization by suitable subtask decompositions and by checking the correctness of the concept during designing. Figure 2.7 shows the system model of an engineer while conceptual designing.

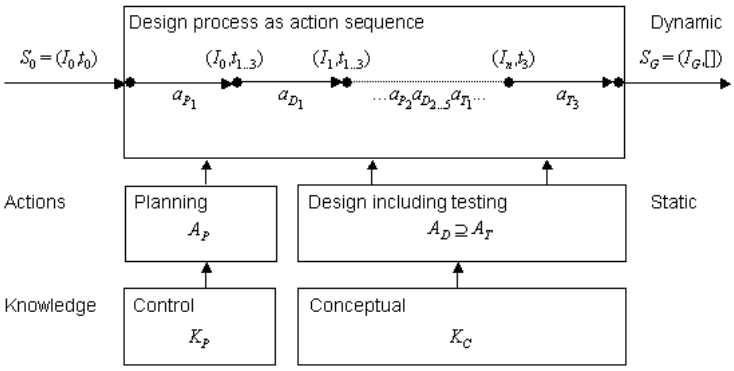


Figure 2.7: System model of the engineer while designing

A typical conceptual design decomposes by planning actions A_P into a known set of subtasks T at lower task levels until the conceptual level is reached. This decomposition was found across different design studies [Eis00, Eis01a, ES02b]. On this level, the engineer develops in a simple subtask the systems or elements of a structural concept by a sequence of design actions and tests it for correctness. Actions A_T determine when a subtask is considered to be solved and when a part of the concept is inconsistent that needs be modified.

At the highest task level, t_0 = design structural concept decomposes into three compound subtasks: t_1 = enter building, t_2 = check feasibility, and t_3 = make structural concept. They have to performed in this order, denoted $t_1 \prec t_2 \prec t_3$. The first task results in the problem specification on which the engineer checks the feasibility to design a concept for the given load and design constraints. On the next lower task level for example, the subtasks of t_3 are: t_4 = design vertical system, t_5 = design slab system, t_6 = size elements, t_7 = design stability, and t_8 = design foundation. The following ordering constraints exist among the subtasks: $t_{4-5} \prec t_6 \prec t_7 \prec t_8$. However, the lateral elements have to be chosen before t_{4-5}. The main subtasks t_4 and t_5 have to be solved in an appropriate order depending on the architectural constraints and loading conditions. If these subtasks are not performed in the right sequence or if the stability system must be changed late in the design process, it might become necessary to redesign some structural systems as discussed in example 3. Dependencies among the subtasks and when they should be performed represent control knowledge. It reduces the number of iterations. Information required for a task to be performed and how it is decomposed into subtasks at a lower level is encoded as precondition. A precondition is evaluated against the current design state, an intermediate solution, to determine if a task and how it is decomposable. A precondition can hence determine which kind of task decomposition is suitable for a developing structural concept.

On the conceptual level, simple subtasks as t_7 = calculate lability or t_4 = design vertical system are solved by design action sequences. For t_7, only the action to calculate lability numbers is required, whereas for t_4 a sequence of actions, like e.g make column and make wall, develop the vertical load-bearing system. At the end of the vertical system design, the engineer checks if the system is complete by a test action. If it is the case, he continues with the next subtask, otherwise he completes these parts of the vertical system that need additional supporting elements to provide sufficient load transfer. Selection of suitable action sequences for a simple subtask are also encoded by preconditions in design actions. In this way, the engineer can sequence design actions to reduce iterations in simple subtasks. A consistent concept that might be still incomplete is the basis for his design. During design, the engineer detects by conceptual knowledge these elements that do not fulfill requirements stipulated in the DIN 1045-1 and modifies them immediately before continuing.

On the basis of the so far acquired information in this chapter, a model of conceptual design for the domain of concrete building structures is formulated.

2.6 Model of Conceptual Design

In conceptual design, the engineer first plans his design approach on the task level before he designs the structural concept at the conceptual level. At the task level, design actions are related to simple subtasks for which their applicability to intermediate solutions is determined by their preconditions. Thus, these two levels are connected via the structural concept under development and the simple tasks. The initial concept can be seen as the starting point of the actual conceptual design because the required information is simply looked-up and copied from the room and occupancy concept, the design brief, and the drawings. The output of conceptual design is a *correct* refinement of the initial concept with:

- a global load transfer and sufficient stability (completeness),
- elements fulfilling the requirements stipulated in the DIN 1045-1 and satisfying the architectural constraints (consistency).

The refinement is carried out in a focused manner due the task level. Refinements are the addition of elements and their composition into structural systems, their arrangement by relations, and the assignment of element/system parameters or state variables to values. As a consequence, conceptual design can be formulated as the problem to refine a given initial interpretation of a structure on the conceptual level to a goal interpretation found in sketches [Bas00]. An *interpretation* is a mathematical structure, which is called a *model* if it satisfies all facts in the knowledge base, which describe the required properties for a fixed signature[8] [TW02]. In the case of conceptual structural design, the properties are *completeness* and *consistency*. Facts in the knowledge base are the initial concept as problem specification in form of loads, constraints, and existing elements, the conceptual knowledge, and the refinement of the initial concept as the solution. What a correct solution for a problem specification is determines thus the conceptual knowledge. The set of models that correspond to correct solutions are goal interpretations that are uniquely described by the interpreted facts defined in some language and assumed to be true [EFT96]. The goal interpretation possesses an extended domain of discourse with additional structural objects and a refined structure because the initial interpretation is not complete. Since the initial and the set of goal interpretations are ordinarily intensionally defined by conceptual knowledge represented by facts in the knowledge base they have to satisfy, I

[8] The initial and the goal interpretation share the same signature.

only define the main characteristics of the required design input and output as a mathematical structure. They are load transfer, system composition, and approximate element design.

Definition 1 (Initial structural concept) *An* initial structural concept *is a structure*[9] $\mathcal{I}_0 = \langle \Delta_{\mathcal{I}_0}, R^{\mathcal{I}_0}, F^{\mathcal{I}_0} \rangle$ *that consists of a non-empty set* $\Delta_{\mathcal{I}_0}$ *(the domain of concrete building structures), the relations* $R^{\mathcal{I}_0}$*:*

- $\textit{structural-object}^{\mathcal{I}_0}, \textit{constraint}^{\mathcal{I}_0}, \textit{system}^{\mathcal{I}_0} \subseteq \Delta_{\mathcal{I}_0}$,
- $\textit{load}^{\mathcal{I}_0} \subseteq \textit{structural-object}^{\mathcal{I}_0}$ *and all* $x \in \textit{load}^{\mathcal{I}_0}$ *maximal in* $\textit{structural-object}^{\mathcal{I}_0}$,
- $\textit{element}^{\mathcal{I}_0} \subseteq \textit{structural-object}^{\mathcal{I}_0}$,
- $\textit{storey-system}^{\mathcal{I}_0} \subseteq \textit{system}^{\mathcal{I}_0}$,
- $\textit{designable}^{\mathcal{I}_0} = \emptyset$ *and* $\textit{stable}^{\mathcal{I}_0} = \emptyset$,
- $\textit{supports}^{\mathcal{I}_0} \subseteq \textit{element}^{\mathcal{I}_0} \times \textit{structural-object}^{\mathcal{I}_0}$,
- $\textit{hasPart}^{\mathcal{I}_0} \subseteq \textit{system}^{\mathcal{I}_0} \times (\textit{element}^{\mathcal{I}_0} \cup \textit{system}^{\mathcal{I}_0})$,
- $\textit{hasConstraint}^{\mathcal{I}_0} \subseteq (\textit{system}^{\mathcal{I}_0} \cup \textit{element}^{\mathcal{I}_0}) \times \textit{constraint}^{\mathcal{I}_0}$,

where $\textit{constraint}^{\mathcal{I}_0}, \textit{structural-object}^{\mathcal{I}_0}, \textit{system}^{\mathcal{I}_0}$ *are pairwise disjoint, and the partial functions* $F^{\mathcal{I}_0}$*:*

- $\textit{state-variable}^{\mathcal{I}_0} : \textit{element}^{\mathcal{I}_0} \to \mathbb{R}$, $\textit{parameter}^{\mathcal{I}_0} : \textit{element}^{\mathcal{I}_0} \to \mathbb{R}$.

The load set might include wind, snow, self-weights and occupancies, and the element set walls, columns, slabs, cores, shear walls, and foundation elements. The constraints are of abstract type like budget, flexibility and allowable vertical load bearing elements on certain storeys. Complex constraints that arise due to the building geometry are disregarded because their satisfaction is not in the scope of support. Since the building height might determine the allowable heights of slab and slab beam elements it is informally considered in the design procedures for estimating element gross sections. Design requirements for the structural elements are ultimate limit and serviceability state constraints or limitations for minimal reinforcement ratios and gross sections. If the respective design requirements are fulfilled, an element is added to the subset *designable*. If a stability system satisfies the stability condition

[9] a non-empty set with relations and functions defined over it

of the DIN 1045-1, it is assumed to be *stable*. A structural goal concept is then defined as a refinement of the initial concept where unchanged sets are left out for brevity.

Definition 2 (Structural goal concept) *A* structural goal concept *is a structure* $\mathcal{I}_G = \langle \Delta_{\mathcal{I}_G}, R^{\mathcal{I}_G}, F^{\mathcal{I}_G} \rangle$ *that is a refinement of* $\mathcal{I}_0$ *and consists of a set* $\Delta_{\mathcal{I}_G} \supseteq \Delta_{\mathcal{I}_0}$*, the functions* $F^{\mathcal{I}_G} = F^{\mathcal{I}_0}$*, and the extended relations* $R^{\mathcal{I}_G}$*:*

- $element^{\mathcal{I}_G} \supseteq element^{\mathcal{I}_0}$,
- $footing^{\mathcal{I}_G} \subseteq element^{\mathcal{I}_G}$ *and only* $x \in footing^{\mathcal{I}_G}$ *minimal in* $structural\text{-}object^{\mathcal{I}_G}$,
- $system^{\mathcal{I}_G} = \{S_V, S_L, S_S, \ldots\} \supseteq system^{\mathcal{I}_0}$ *with* S_V, S_L, S_S *the vertical, the lateral and the slab system,*
- $designable^{\mathcal{I}_G} = element^{\mathcal{I}_G}$,$stable^{\mathcal{I}_G} = \{S_L\}$,
- $supports^{\mathcal{I}_G} \supseteq supports^{\mathcal{I}_0}$ *that is a total order over* $structural\text{-}object^{\mathcal{I}_G}$,
- $state\text{-}variable^{\mathcal{I}_0}(x) \leq parameter^{\mathcal{I}_0}(x)$ *as element requirements.*

An intermediate *design state*, denoted $\mathcal{I}$ is a partial refinement of $\mathcal{I}_0$ that has been constructed by a sequence of design actions, and does not possess the goal structure. The *design state space*, denoted $\mathcal{W}$, can be hence conceived as the set of interpretations reachable from $\mathcal{I}_0$ by instantiated actions of the action types A_D. In the structure, the function F is represented by the predicates $designable^{\mathcal{I}}$ and $stable^{\mathcal{I}}$ for elements and the lateral system. The expected behavior B_e is given by the equivalence classes of domain objects, the total order relation, and the inequality relation. The structure S of the concept is described by objects $\Delta_{\mathcal{I}} \setminus constraint^{\mathcal{I}}$, the relations $hasPart^{\mathcal{I}}$ and $supports^{\mathcal{I}}$, and the structural variable $parameter^{\mathcal{I}}(x)$. The actual behavior B_s corresponds to assigned values for $state{-}variable^{\mathcal{I}}(x)$ and related elements, loads, and systems through $supports^{\mathcal{I}}$. The support relation is twofold employed because of missing detailed information in conceptual design. On the former definitions, the problem of conceptual design in the domain of concrete building structures is defined.

Definition 3 (Conceptual design problem) Conceptual design *is the* problem *to refine an initial concept* $\mathcal{I}_0$ *to a structural goal concept* $\mathcal{I}_G$*.*

The *set of goal states* $\mathcal{W}_G$ can be understood as the class of interpretations that possess the same structure as $\mathcal{I}_G$ [EFT96, TW02].

Example 7 (Small conceptual design problem) Consider example 2, which represents a part of a larger conceptual design for which no constraints were imposed and the foundation and stability system do not have to be designed. The subtask is to transfer the applied load through the beam to supporting elements and size it for moment capacity. The conceptual design problem is to find for the initial structural concept $\mathcal{I}_0 = \langle \Delta_{\mathcal{I}_0}, R^{\mathcal{I}_0}, F^{\mathcal{I}_0} \rangle$ with:

- $\Delta_{\mathcal{I}_0} = \{P_{\text{Ed}}, B_1\}$,
- $\textit{structural-object}^{\mathcal{I}_0} = \{P_{\text{Ed}}, B_1\}$, $\textit{constraint}^{\mathcal{I}_0} = \emptyset$, $\textit{system}^{\mathcal{I}_0} = \emptyset$,
- $\textit{load}^{\mathcal{I}_0} = \{P_{\text{Ed}}\}$, $\textit{element}^{\mathcal{I}_0} = \{B_1\}$,
- $\textit{designable}^{\mathcal{I}_0} = \emptyset$,
- $\textit{supports}^{\mathcal{I}_0} = \{(B_1, P_{\text{Ed}})\}$, $\textit{hasPart}^{\mathcal{I}_0} = \{(S_V, B_1)\}$,
- $\textit{hasConstraint}^{\mathcal{I}_0} = \emptyset$, and
- $\textit{state-variable}^{\mathcal{I}_0}(P_{\text{Ed}}) = 100\text{kN}$,

a vertical load-bearing system as goal concept $\mathcal{I}_G = \langle \Delta_{\mathcal{I}_G}, R^{\mathcal{I}_G}, F^{\mathcal{I}_G} \rangle$ with:

- $\Delta_{\mathcal{I}_G} = \{P_{\text{Ed}}, B_1, C_1, C_2\}$,
- $\textit{structural-object}^{\mathcal{I}_0} = \{P_{\text{Ed}}, B_1, C_1, C_2\}$ with P_{Ed} maximal and C_1, C_2 minimal in the set,
- $\textit{system}^{\mathcal{I}_G} = \{S_V\}$,
- $\textit{load}^{\mathcal{I}_G} = \textit{load}^{\mathcal{I}_0}$, $\textit{element}^{\mathcal{I}_G} = \{B_1, C_1, C_2\}$,
- $\textit{designable}^{\mathcal{I}_G} = \{B_1\}$,
- $\textit{supports}^{\mathcal{I}_G} = \textit{supports}^{\mathcal{I}_0} \cup \{(C_1, B_1), (C_2, B_1)\}$,
- $\textit{hasPart}^{\mathcal{I}_G} = \textit{hasPart}^{\mathcal{I}_0} \cup \{(S_V, C_1), (S_V, C_2)\}$,
- $\textit{state-variable}^{\mathcal{I}_G}(M_{\text{Ed}}) = 200\text{kNm} \leq \textit{parameter}^{\mathcal{I}_G}(M_{\text{Rd}}) = 200\text{kNm}$.

This problem is often called *abductive design* [RN95, PMG98, RS03], *model construction* by BUCHHEIT and *et al.* [BKN95, BBH94], or *model assignment* by NEBEL [Neb01a] when possible solutions are defined by a set of facts in a knowledge base for which a model as solution is constructed. In these frameworks, knowledge about how elements of the domain can be

related and values can be assigned for them is thus not extensionally defined by enumerating all tuples in the relations but intensionally by describing the required property of possible solutions by facts in some logical language [GRS00, RN95]. However, the question what kind of structure a solution possesses is open. The model construction view also subsumes that design is represented as constraint satisfaction problem according to HÖLLDOBLER [Höl01] for which constants representing domain objects have to be assigned that satisfy the domain constraints.

On the underlying model construction approach, the design of unknown structural concepts with new relations and functions in the structure is not possible because the conceptual knowledge fixes the structure of possible solutions. The engineer has to search a structural concept, which is not explicitly known beforehand but can be deduced from implicit conceptual knowledge. Therefore, conceptual design of building structures is similar to *configuration design problem solving*[10] [WS97] for which knowledge-based methods have been proposed. Because the search space tends to be quite large due to possible combinations of components by their connections and parameter value assignments, some of these methods draw also on the notion of control knowledge to find a solution efficiently. Control knowledge represents suitable action sequences to construct a correct configuration for a problem specification. The methods search for reasonable action sequences to construct correct configuration instead of searching directly in the state space of configurations. Since the state space of possible structural concepts is large, too, and the analyzed design studies indicated such a design approach, the model of conceptual design should incorporate the idea of control knowledge. It is represented by a suitable task decomposition of conceptual design. NEBEL even states that the model formulation should incorporate the method practitioners use to solve the problem because the right model makes the solution of the problem tractable by computers [Neb96]. I will return to configuration design in Chapter 3 in which I will review different methods for configuration design that motivated my choice for the description logic based planning formalism on which the prototype system operates.

State transition systems are useful to model dynamical systems as the engineer while designing since effects of actions are discrete and fully predictable

[10] A configuration design problem is to construct a correct configuration from a fixed, predefined set of components connected by ports for a given specification, which comprises of constraints and a partial incomplete configuration. The correct configuration consists of a set of parameterized components and a description of the connections between the components by ports satisfying given functional requirements and constraints imposed on the port connections [MF89, Bro97].

[Wun00]. It is based on the idea of NEWELL and SIMON that a dynamic system with an initial state can be described in terms of its actions to change the initial state into successor states for which some of them are goal states in which the system halts [NS72]. In my case, a system state is a structural maybe incomplete concept together with a list of remaining tasks to transform the current concept into a correct one. Subsequent states are reached by pure sequential actions of task decomposition, concept manipulation, and its testing. This idea is very similar to the common definition of a *deterministic finite automata* except that for these the transition function is usually defined to be partial [Sch99]. Such a system is ordinarily used to describe state models for planning domains for which the problem is to find an action sequence that generates a state trajectory from an initial to a goal state [BG01, Rin03]. Because such a state transition system must model action sequences that correspond to context free sets that result from cycles in subtasks [EHN95, Ero95], the state transition system is extended by a storage for a sequence of subtasks [GNT03]. It works like a stack of a *pushdown automata* [HU94] except that a solution is a correct structural concept with no remaining subtasks on the stack. Tasks can be pushed onto the stack or removed from the top of the stack. An action sequence $[a_1, \ldots, a_n]$ transforms the *initial system state* $\mathcal{S}_0 = (\mathcal{I}_0, t_0)$ into a *goal system state* $\mathcal{S}_G = (\mathcal{I}_G, [])$ where [] denotes the empty stack. $\mathcal{S} = \mathcal{W} \times T^*$ is a *set of system states* reachable by actions sequences over A from $\mathcal{S}_0$. Furthermore, the additional system parameter K of conceptual and control knowledge, denoted K_C and K_P, is introduced into the system because it affects the system behavior. An action $a(K)$ depends thus also on the unchanged system parameter of knowledge during designing. The engineer can be described by a *system state* of the design process $s \in \mathcal{W} \times T^*$ with $(\mathcal{I}, [t_1, \ldots, t_n])$ which includes the current design state, and the remaining subtasks at some task level. A *transition function* $\delta : \mathcal{W} \times T \times A(K) \rightarrow \mathcal{W} \times T^*$ maps each system state by a design step to a new one for which only the concept $\mathcal{I} \in \mathcal{W}$ and the tasklist $t \in T^*$ (T^* the set of subtask sequences over T) are changed by state transitions. On this basis, the engineer can be modeled as dynamical state system during conceptual design.

Definition 4 (State model of conceptual design) *A state model of* conceptual design *is a system* $M_{CD} = \langle \mathcal{S}_0, \mathcal{S}, A(K), T, \mathcal{G}, \delta \rangle$ *with:*

- $\mathcal{S}_0 = (\mathcal{I}_0, t_0)$ *the initial system state,*
- $\mathcal{S} = \mathcal{W} \times T^*$ *the set of system states reachable from* $\mathcal{S}_0$ *by* $[a_1, \ldots, a_k]$,

- $A(K) = A_D(K_C) \sqcup A_P(K_P)$ *the set of design actions including test actions* A_T *and planning actions,*
- $T = T_S \sqcup T_C$ *the set of simple and compound subtasks,*
- $\mathcal{G} = \bigcup_{\mathcal{I} \in \mathcal{W}_G} (\mathcal{I}, []) \subseteq \mathcal{S}$ *the set of goal system states.*

The transition function δ *is given for the different action types as:*

- $\delta(\mathcal{I}, t_S, a_D) = (\mathcal{I}', t_S)$,
- $\delta(\mathcal{I}, t_C, a_P) = (\mathcal{I}, [t_1, \ldots, t_n] \in T^*)$,
- $\delta(\mathcal{I}, t_S, a_{T_1}) = \begin{cases} (\mathcal{I}, []) & \text{if } \mathcal{I} \text{ is complete} \\ (\mathcal{I}, t_S) & \text{else,} \end{cases}$
- $\delta(\mathcal{I}, t_{S_1}, a_{T_2}) = \begin{cases} (\mathcal{I}, t_{S_1}) & \text{if } \mathcal{I} \text{ is consistent} \\ (\mathcal{I}, [t_{S_2}, t_{S_1}]) & \text{else with } t_{S_2} \text{ a simple modification task.} \end{cases}$

Intuitively, the transition for a design action a_D means that the engineer changes the concept but not the subtask. The transition for the planning action a_P means that the engineer sequences the subtasks by task decomposition on the next lower task level. The transition for the test action a_{T_1} depends on the completeness of the design state in terms of load transfer. If the engineer has designed a complete structural system or element in a simple subtask, the transition for the test action removes the subtask from the top of the stack, denoted [], whereas in the other case the simple subtask remains on the stack and incomplete elements are marked by the system. The transition for the test action a_{T_2} depends on the correctness of the design state. If the engineer has designed a correct structural system or element in a subtask, a transition for a test action keeps the subtask at the top of the stack, whereas in the other case it pushes a simple subtask on the stack to modify the incorrect element. The initial concept $\mathcal{I}_0$ is problem instance specific assigned by the engineer on the given drawings and information. The actions A, the tasks T, and the knowledge K are persistent, which the engineer retrieves from long-term memory during designing. On this chosen model, a solution does thus not only include a correct structural concept $\mathcal{S}_G$ but also the design action sequence $[a_1, \ldots, a_n]$ for refining the initial concept into a correct one. For a system M_{CD}, a design step is a binary relation $(\mathcal{I}, [a : x], [t : y]) \vdash (\mathcal{I}', x, [z : y])$ over the set of system states if $(\mathcal{I}, z)$ in $\delta(\mathcal{I}', a, t)$. The relation $\vdash^*$ is the reflexive and transitive closure of $\vdash$ resulting from the successive application of the transition function δ.

Definition 5 (Solution to the model of conceptual design) *Let M_{CD} be a model of conceptual design. $\mathcal{S}_G \in \mathcal{G}$ and $\mathcal{P} = [a_1, \ldots, a_n] \in A^*$ are a* solution *for M_{CD} if $(\mathcal{S}_0, \mathcal{P}) \vdash^* \mathcal{S}_G$.*

The definition of the engineer as a state transition system and its solution to this model leave some open questions because the state transitions are only defined for types of actions and over the abstract state space. However, the model gives an intuition about how the engineer solves the conceptual design task by different design actions. In Section 5.2, I will compare this model of conceptual designing to the one of HAUSER, who proposed a cognitive architecture, which assigns different reasoning processes to conceptual design problem solving in the routine structural domain [Hau98].

The engineer considered as system can be visualized by a graph in which a set of nodes annotated by stacks are used to describe a system state. They represent given snapshots of the world at different times during designing. Edges connect these nodes indicating possible changes that can be caused by the engineer's actions. Another state is reached by applying an action to a given system state.

Example 8 (State model in conceptual design) Figure 2.8 illustrates the idea of a state model. It represents a fraction of a possible behavior of an engineer in the design process. The model comprises of the seven system states $\mathcal{S}$ as depicted, which are described by a design state and a current tasklist. The initial system state is $\mathcal{S}_0 = (\mathcal{I}_0, t_1)$ and the possible design actions are $A = \{a_P, a_{D_1}, a_{D_2}, a_{D_3}, a_{T_1}\}$ to change a system state. The subtasks $T = \{t_2\}$ include only one simple task. The set of goal system states are $\mathcal{G} = \{(\mathcal{I}_1, []), (\mathcal{I}_2, [])\} \subseteq \mathcal{S}$. The transition function δ is given as shown in figure 2.8.

For example, the initial process state $(\mathcal{I}_0, t_1)$ might include a slab representation and the task to design a column as support for it. Then the planning action a_P decomposes the task t_1 into the subtask t_2 which can be solved by suitable design actions. The action a_{D_1} is used to insert a column and to establish the support relation between the column and the slab. Now, the engineer tests by the action a_{T_1} if this support relation is sufficient. If it is the case, the engineer has reached a goal state. Otherwise, he has to modify the column design because the subtask t_2 remains on the stack. In the example, he removes the new column an starts the subtask t_2 at the beginning. The engineer might add another support relation by design action a_{D_2}. Now, he tests the completeness of the solution by the action a_{T_1}, which is in this example an element of the goal set. The paths which transform

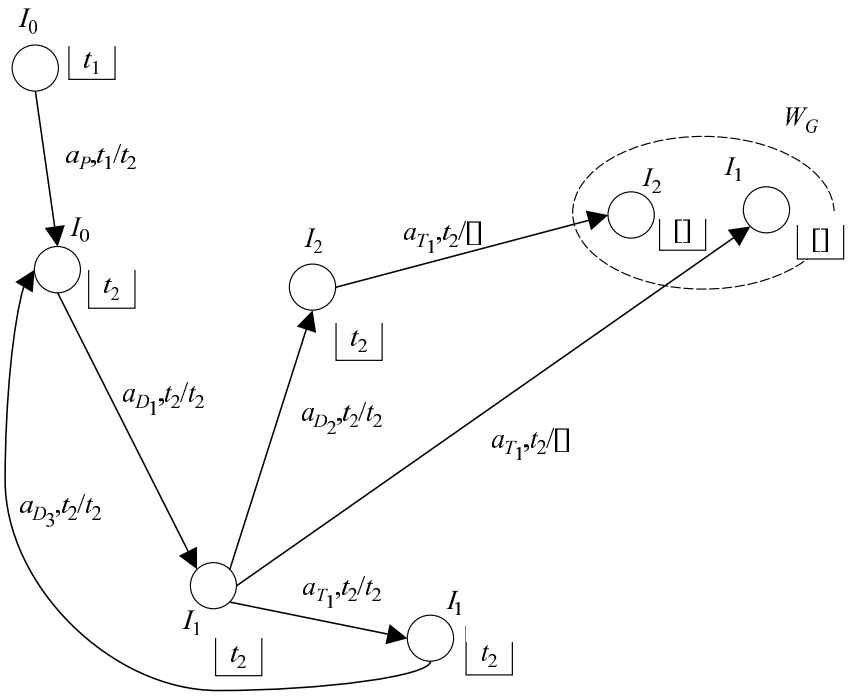


Figure 2.8: A simple state model

the initial process state in either of the two goal states represent a solution, e.g. $\mathcal{P} = [a_P, a_{D_1}, a_{D_2}, a_{T_1}]$. To avoid cycles in simple tasks preconditions in design actions encode local control knowledge. In case of a column design, the engineer had checked its gross sectional requirements by an action a_{T_2} that had pushed a simple modification task on the stack for an inconsistent column.

Goal states are not explicitly enumerated but described by conceptual knowledge about correct and complete concepts in a description logic language. Design states are given by state descriptions in this language, too. Possible actions and control knowledge are defined in a hierarchical planning language that operate on these state descriptions. Reasoning services for task decomposition, action selection, consistency testing and completeness testing will be implemented by the formalism in the next chapter.

Chapter 3

The Formalism

First, a brief introduction about knowledge representation is given. Thereafter, different knowledge-based methods that support configuration design and share similarities with the conceptual design problem are discussed. Since these computational paradigms do not completely cover the envisaged design support for the model of conceptual design, I introduce the selection of a description logic system combined with a superimposed hierarchical planning. I will discuss my specific choice of the expressive description logic $\mathcal{ALCQI}(\mathcal{D})^-$ with the system RACER and the hierarchical task planning method SHOP to implement the interactive formalism for the computer-aided design system. At the end of the chapter, I give a detailed presentation of the description logic based planning formalism.

3.1 Design Knowledge Representation

Conceptual design is a complex intellectual activity that cannot fully be mapped onto a computational process. Therefore, a couple of assumptions were made in Chapter 2, which resulted in definition 4 of model for conceptual design. In this model, the engineer searches the system state space along the task decomposition for a goal concept by actions. The search is conducted by interacting with the environment or the design knowledge, e.g. sketching the concept or retrieving known facts, also called *believes*, from long-term memory. Because I assume that the design environment and knowledge are to be either true or false I exclude to model the engineer's believe state, intentions, etc. [NG89]. In general, the engineer is interested to derive implicit believes from a collection of known facts in a logically sound and complete

way. Because the real design objects are not present at design time the engineer applies design symbols for representing a set of believed propositions. They are part of his conceptual model about the structure on which actions operate and control knowledge can be encoded. Knowledge must be declarative and independent of its usage since it must be versatilely applicable in many situations [Neb01a]. The engineer has to reason because many beliefs, made up of represented symbols, are only implicitly given or created during the course of design. The formal manipulation of design symbols by actions, representing the structural concept during design, allows to produce new propositions. This behavior might be called *intelligence*[1] [PMG98]. *Knowledge-based systems* realize this behavior on computers (see for example [Neb90, BHS93] for reasoning and knowledge representation techniques and systems).

Approaches to knowledge representation divide roughly into logic-based formalisms and other, non-logical representations. The former developed out of the idea that first-order predicate logic (henceforth FOL) can be used to represent facts about the world, the latter emerged from the idea to base representations on cognitive notions - for example a frame and a rule. In a logic-based approach except of non-monotonic reasoning, the representation language is usually a variant or subset of FOL and the reasoning amounts to the verification of logical consequences or testing if a theory is consistent. In the non-logical approach, reasoning is accomplished by ad-hoc procedures that manipulate knowledge represented by means of data structures. This results in a lack of precise semantic characterization for the reasoning processes, often called *services*. This lack leads to responses of a knowledge-based system that are not traceable for the user.

Formalisms, also termed *computational methods*, can be used to map the reasoning involved in design onto the computer to support the engineer during his work [CRR$^+$90, Cha90, TS92]. One class of methods, introduced informally in the previous chapter under the term "configuration", has been developed in the field of Artificial Intelligence. In the next section, I will first describe configuration design in more detail to compare it to the model of conceptual design. On this basis, I will be able to discuss the applicability of proposed configuration-design-methods. They shall be used to implement the model of conceptual design and thereby support the solution process.

[1] McCarthy [MH69] calls a machine intelligent if it solves a class of problems requiring intelligence in humans.

3.2 Methods for Configuration Design

Methods of *configuration design* are employed to reduce design costs, error rates, and automate the development stage in the design process partially. According to GERO and CHANDRASEKARAN and BROWN [Ger90, BC89], which categorize design problems into *design problem classes*, configuration design can be roughly classified as a class 3 design problem. They assume for configuration design that the product structure with all its components, relations, and parameters is known by experience of similar design projects. A global solution can be top-down constructed from partial solutions and components. NAVINCHANDRA states for example two important characteristics, which are not supported by methods for solving a class 3 design problem: the possibility of redesigning partial configurations and the exploration of the design state space by constructing new solution alternatives during designing [Nav91]. On these limitations, configuration design can be seen as a special case of design activity, which obeys three key features: the artifact being developed is assembled of a fixed set of well-defined components, the artifact has a nearly fixed structure, and components interact with each other in predefined ways [Bro97, WS97]. According to [MF89, SW98, GK99] the problem of configuration design can be described as follows: Given an initial partial configuration, conceptual knowledge about components with their parameters and their relations (taxonomical, compositional), a set of constraints about possible component connections, a set of configuration actions, and control knowledge about the configuration process, construct a complete configuration by a sequence of configuration actions satisfying the constraints.

Depending on how the components, the assembly structure and constraints have been defined prior to the configuration task, configuration design can be seen to span from local to full configuration design [WS97]. Many others, see for example [GK99, TS92], have pointed out that methods from configuration design face difficulties when their application is extended to assisting practical design problems as for the model of conceptual design. There, the function of the artifact has to be taken into account to compute a correct configuration.

In the following, I will introduce methods for solving the configuration design problem and discuss their limitations to develop my envisaged design support. I call the methods *model-based* where model-based refers to a conceptual model of the configuration, which is necessary for the desired interactive conceptual design support[2]. I discuss the methods in terms of offered

[2]For a complete list of configuration methods see for example [GK99, SW98, WS97] or the special issues Vol. 12, No. 4, 1998 and Vol.17, No.1, 2003 about configuration of the Journal Artificial Intelligence for Engineering Design, Analysis and Manufacturing.

reasoning services to test the formal correctness of the obtained concept and to allow bottom-up design. The correctness is important because engineers expect especially from a computer-aided conceptual design system a reliable support. It depends on the soundness and completeness of the employed reasoning services, which process the language to draw inferences on the knowledge. Higher expressiveness of a language leads to an increased complexity for solving the inference problems. At some point of expressiveness, a service becomes incorrect because it misses for example some inferences. Therefore, there is a trade-off between expressiveness and tractability, determining the correctness, as mentioned by LEVESQUE and BRACHMAN [LB87]. If the language is highly expressive as FOL, the inference problems become semi-decidable. Thus, only limited reasoning support can be retained since for some cases the inference machine runs forever or gives the wrong answer [Baa99].

The model-based methods can be roughly separated into *structure-based*, *graph grammar-based*, *constraint-based* and *logic-based*. After their discussion, I will discuss my choice for a description logic based planning formalism to implement the model of conceptual design.

The configuration system ENGCON uses a *structure-based* method [HWG00]. It represents conceptual knowledge in the cyclic frame language BHIBS developed by CUNIS [Cun92], which allows to reason about taxonomical and compositional relations of the developing configuration. The user can top-down construct a goal configuration along is-a and has-part relations defined in the domain taxonomy. A dynamic realization of the configuration and constraint propagation in a numerical constraint network support the user during the configuration process. Furthermore, control knowledge is encoded by an independent agenda-based mechanism [Gün89]. It enables the refinement of the configuration by selecting a configuration action, called agenda entry in this method. For the entry, different calculation methods are selectable. The configuration process proceeds until the agenda is empty. Thus, the engineer can explore the configuration state space by an ordered sequence of actions. The system allows backtracking to previous states and the evaluation of the current configuration state for numerical and simple relational constraints in data logic [Kre02]. The frame language employed is undecidable, which SCHRÖDER *et al.* showed by translating BHIBS into the description logic $\mathcal{ALCQIOD}$, where the letters stand for different language constructors of the description logic [SML96].

The UPGRADE system employs a *graph grammar* method for supporting the user in detecting inconsistencies in architectural building layouts at the conceptual design stage. The graph grammar approach focuses on the design support by conceptual knowledge. Kraft *et al.* [KMN02] use the PROGRES language for specifying the conceptual knowledge. This graph language is very expressive and allows the representation of conceptual knowledge and reasoning about labeled cyclic graphs. The solution is represented by an instance graph, which can be designed from a type graph. Design actions are represented by rewriting rules for specific nodes or subgraphs from the instance graph. A similar approach is taken for example by STEIN *et al.* [SS01] in the conceptual design domain of chemical engineering. They define graph transformation rules over a labeled-graph as design actions in order to develop a goal configuration where they develop a structure model corresponding to a configuration. These approaches do not take control knowledge about the suitable sequence of subtasks and design actions into account. However, this knowledge is important to reduce iterations in the design process.

A *constraint-based* method for supporting the conceptual design of a bicycle structure from a function-means tree is investigated by SULLIVAN [O'S02]. The underlying design language GALILEO is an expressive constraint programming language based on FOL enriched with frame information and concrete domains. The configuration problem is processed by a method that combines an arc-consistency procedure and constraint filtering. The design process is structured according to a model of conceptual design, which includes the subtasks: function specification, schemes generation, and evaluation and comparison of schemes. After the function has been defined, the function is decomposed into embodiments, which in turn have to fulfill certain subfunctions. An explicit representation of behavior is not given but compiled into a function-embodiment mapping, whereas control knowledge is represented by function-embodiment pairs. Local concrete valued behavioral constraints restrict furthermore the possible set of solutions. The engineer can thus assign in which sequence he embodies the functional requirements, whereas the system checks the schemes' correctness on the conceptual knowledge. The problem of computing the correctness of a scheme is undecidable because of the design language's expressiveness. LOWE *et al.* [LPB98] use a similar method, which is enhanced by representing control knowledge on a separate declarative level and process it by a proof planning procedure. They propose their system for improving the maintainability of knowledge bases for configuration design.

Logic-based methods can be roughly separated into FOL and description logic (abbreviated DL) methods. Description logics are a subset of FOL [Bor96]. They offer additional reasoning services and remain decidable [BCM+02].

FRIEDRICH *et al.* [FS99] solve automatically the configuration problem with a FOL theorem prover. They employ the unsatisfiability service for testing the consistency of a current configuration. Due to the very high expressiveness, testing for satisfiability in FOL is undecidable and provides thus limited reliable design support. In addition, they use some form of closed-world model checking ensuring the validity of a complete configuration at the end of a configuration session. Configuration actions and control knowledge about the organization of the configuration task are not represented.

The configuration system PROSE used the description logic $\mathcal{ALNFIH}$ plus facilities for dealing with numbers to provide interactive support during configuring [MW98, BCM+02]. Conceptual knowledge about admissible configurations is represented by an unfolded concept definition under which the partial configuration is realized by stating new facts about components. A set of rules is applied for inferring new facts triggered by the user's input. A sound and complete so-called ABox reasoning service for instance checking serves for configuring admissible parts of a solution and deciding rule applicability. Because a rule application can lead to inconsistent configuration states they provide a backtracking mechanism.

BUCHHEIT and BAADER pursue another approach to solve automatically the configuration problem with a DL reasoner for the language $\mathcal{ALCQ}$ with cardinal restrictions on concepts [BKN95, BBH94]. They propose to employ a consistency service for the generation of a goal configuration from an initial partial configuration. Conceptual knowledge is also represented in a so-called TBox for which, while testing the initial configuration's consistency, a model can be constructed as goal configuration.

WEIDA introduces a *closed taxonomy assumption* for solving the configuration problem by a DL reasoner [Wei96]. It allows the reasoner to infer more information in comparison to the open world assumption because the extension of the taxonomy by concepts created during the solution process is forbidden. By this assumption, an exploratory component while configuring is excluded but the complexity of the inference problem is strongly decreased.

To find a solution for the model of conceptual design is not supported by a single method from above. Conceptual design is action-based and demands the representation of the global load transfer. They govern the final structural concept. Furthermore, the taken design approach realizes main structural systems by relating structural elements through support conditions in a

bottom-up synthesis process for known simple subtasks. This type of solution approach usually separates design from pure configuration tasks [Bro97]. WIELENGA and SCHREIBER call this type *skeletal design* [WS97] based on the work about planning with constraints by STEFIK [Ste81].

The envisaged design support of the system has to provide services for the evaluation of single elements in terms of their load transfer, for testing their consistency on the DIN 1045-1, for dynamic planning of the subtasks, and for design action proposition. I employ therefore an approach, which combines BAADER's and BUCHHEIT's idea of consistency checking with LOWE's planning formalism. For defining the description logic based planning formalism with the corresponding reasoning services, I extend these ideas by two aspects. First, I include completeness testing by local closed world reasoning about single elements since open world reasoning like consistency testing excludes the detection of incomplete elements, e.g. missing support relations. Second, I incorporate the means for interleaving design actions and hierarchical planning by which the structural concept can be developed in a bottom-up fashion. This is done by using the work of NAU *et al.* about hierarchical task decomposition planning [NCLMA01]. It allows the engineer a bottom-up design of the structural concept for planned subtasks. In the next subsection, I will introduce the formalism with its design language and the corresponding reasoning services that implement the testing of a structural concept and the planning of the design process.

3.3 The Description Logic Planner

In order to reason about design knowledge in a computer to support the engineer in conceptual design, knowledge has first to be represented in a language. I employ two different languages for the representation of knowledge and four services to reason about it. The conceptual model including the developing structural concept and the conceptual knowledge about structural systems, elements, and their relational load transfer behavior will be represented in a description logic. I use a so-called *terminological component* to represent the persistent conceptual knowledge and a so-called *assertional component* for the structural concept. I introduce an additional component to represent the load transfer of a structural concept. The DL services are consistency and correctness testing. Control knowledge and actions will be represented by a hierarchical planning language. The planning services are method and operator application. Method application allows to decompose tasks into suitable subtasks by planning actions, whereas operator application selects suitable

design actions to solve simple subtasks. The different knowledge types are integrated via the assertional part of the description logic, which is thus the base language over which parameterized actions including planning actions are defined.

Definition 6 (Design symbols) *Let* L *and* P *be two disjoint sets, where* L *denotes the set of* description logic names *and* P *the* planning names*, respectively. They are used to represent the conceptual design domain for* M_{CD}*.*

A refined system model with the different components is shown in figure 3.1. The control knowledge is represented by methods and actions' preconditions and the conceptual knowledge by so-called TBox and CBox.

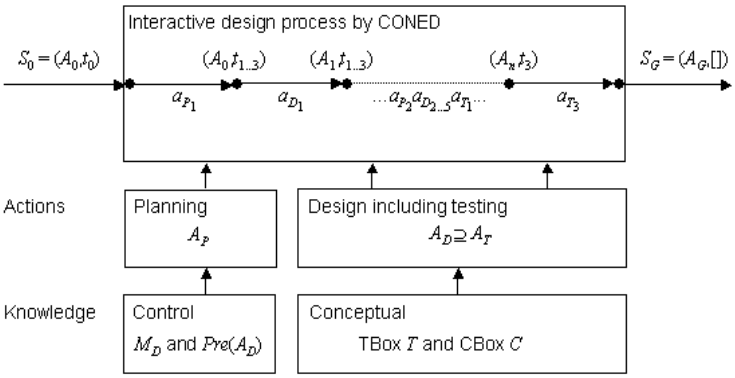


Figure 3.1: System model of the engineer while designing

Because of the mentioned shortcomings of non-logical formalisms in the last subsection, a logic-based approach is taken in this thesis. This approach separates the representation of the design knowledge from the services. It is especially well-suited for the representation of complex, incrementally evolving domains like the one of building structures because the knowledge base can be independently developed from the reasoning services. The logical approach comprises the following steps according to [Neb01b, Baa02] to realize a logic-based formalism like the one of the thesis:

- definitions of a formal language including logical, non-logical symbols, and syntax rules,
- provision of compositional semantics, which interprets the non-logical symbols and defines rules for determining combined interpretations of single symbols,

- definition of reasoning services for the design language that obey the logical entailment relation $\models$ and the transition relation $\vdash$,
- specification of algorithms that implement the reasoning services.

The need of assigning computational tractable techniques to realize the design support on the symbolic processing level led to the choice of a description logic based planning formalism. The next two subsections will introduce description logics and planning. They define first the sublanguages with the corresponding reasoning services, and then the algorithms. The introduction of description logics is taken from [BCM$^+$02] and for planning from [GNT03] for which the examples have been adapted to the conceptual structural design domain.

3.3.1 The Description Logic

Description logics descended form structured inheritance networks to overcome the ambiguities of early semantic networks and frames as used in the KL-ONE system [WS92]. They restrict possible language constructs and result in decidable inference problems. Description logic is a subset of FOL [Bor96] but enhances the formalism by additional reasoning services as for example computing the taxonomy [BHS93]. This service is especially important for complex domains like the one of building structures, where a correct taxonomy has to be built up before deploying the computer-aided conceptual design system. In addition, description logic procedures are sound and complete, and always terminate, which is required for reliable support.

A *graph* can give some intuition about the representation of knowledge in a description logic system. *Nodes* are employed to characterize *concepts* or *objects*, interpreted as sets of individual objects, and *edges* are employed to characterize *roles* that can hold between concepts. They are a characteristic feature of description logics and are interpreted as relations over individual objects. Complex relations and individual objects, which are associated with a concept, are themselves also represented as nodes. Furthermore, concepts can have attached simple attributes. A graph like this is called a *terminology*.

Example 9 (Terminology about a structure) Figure 3.2 shows a terminology, which represents knowledge about the conceptual design domain described by some concepts and roles. It represents the generality/specificity of the involved concepts. For example the edge between Slab and StructuralElement says that slabs are a subset of structural elements and the edge

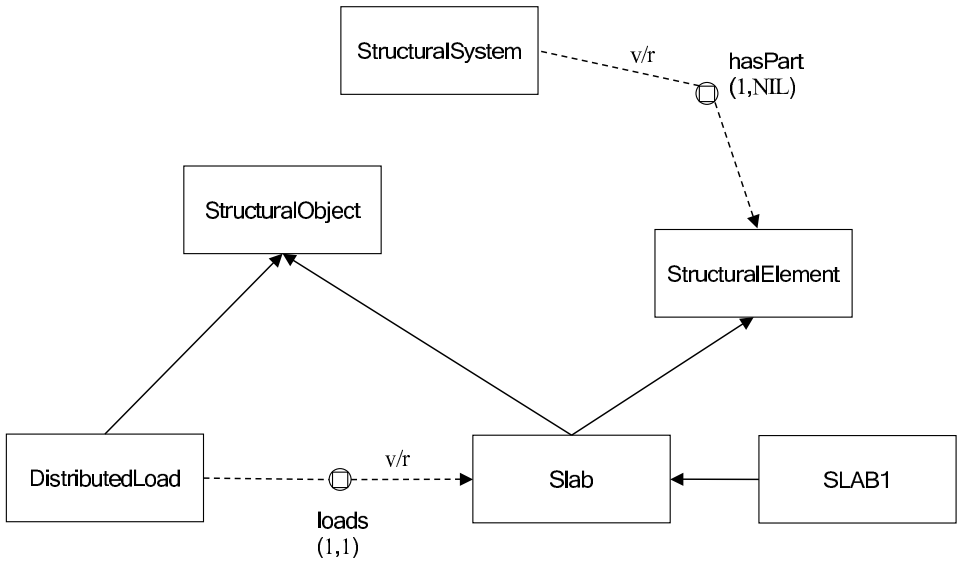


Figure 3.2: An example graph

between SLAB1 and Slab says that it is an object of the concept. The subset relation defines a hierarchy over the concepts. When a concept is more specific than some other concept, it inherits the properties of the more general concept. If for example, the structural element has dimensions, then a slab has dimensions, too. The concept of DistributedLoad has a role loads as property. The role restricts the range of types of objects that can fill that role, denoted by the label v/r. Additionally, the node has a number restriction (1,1), which specifies that one distributed load can only load one slab. The concept StructuralSystem has a value restriction hasPart, which specifies that a system is at least composed of one StructuralElement. Observe that there may be implicit relations between concepts. For example, the concept Slab is a more specific descendant of the concept StructuralElement and as a result would inherit from StructuralElement that it can be part of a StructuralSystem.

It is the task of a description logic system to find such implicit relations among concepts, roles and objects, which might be much more complex. To find propositions implicit in the structure, a system needs to draw *inferences* on it. If the structure becomes more complex, it becomes more difficult for the reasoning system to draw inferences on all components and to give always right answers. Therefore, there is a trade-off between expressiveness of the representation language and the difficulty of reasoning over the representation

built using the language, already mentioned in Section 3.1. The expressiveness is the result of the constructs available in the language for building complex concept and role expressions. I will later discuss the necessary constructs to represent the conceptual design domain, where the goal is to represent all required information while retaining reliable design support. Even though the employed inference problems have a high worst-case complexity, it turned out that they behave well for the conceptual design domain. The closed taxonomy assumption even decreases the complexity during the conceptual solution process because only defined or atomic concepts and atomic roles are offered to the engineer to build up the structural concept.

To process terminologies (graphs) they have to be represented in a language for which a precise characterization of the meaning of the language constructs and the set of inferences drawn on it is needed. Description logics are such a language family because their constructs and reasoning services have a precise meaning while the inferences remain decidable. The specific language is named according to the given set of constructs for building up complex concept and role terms. Concept expressions are variable-free in comparison to FOL expressions. A description logic employs two disjoint alphabets of symbols, which can be used to denote *basic concepts* and *atomic roles*. Basic concepts correspond to unary predicate symbols and basic roles to binary predicate symbols in FOL. The former are employed to express simple properties, the latter to express relationships between concepts. For example, the *intersection of concepts* is denoted $C \sqcap D$. Other concept constructors are *negation* $\neg C$, *conjunction* $C \sqcup D$, *full existential quantification* $\exists R.C$, *value restriction* $\forall R.C$, and *number restriction* of the kind $\exists_{\leq n} C$.

Concept expressions are interpreted as the set of objects, also called *individuals*, that satisfy the properties stated by the expression. Concepts and roles have set-theoretic semantics, which reside in a relational structure, that can be arbitrary and infinite. Atomic concepts are interpreted as subsets of the interpretation domain, also named *universe of discourse*, while the semantics of the other constructs are then specified by the set of individuals denoted by each construct. For example, an intersection $C \sqcap D$ restricts the set of objects to belong to C and D. Another example is the value restriction $\forall R.C$ that requires that all individuals being in the relation R belong to the concept C. In such a way, the meaning of a complex concept expression is composed from primitive ones, which is called *compositional semantics*. Description logics possess furthermore *open world semantics* in comparison to the ordinary semantics of databases. While, a database instance represents exactly one interpretation by the objects and listed tuples, a description logic instance can have many interpretations, namely *models* of the knowledge base. Thus,

absence of information means not negative information but lack of knowledge. This is of special importance in conceptual design for which missing information allows the engineer to develop different models in a flexible way. A computer-aided design system has to take this requirement into account because it must not consider missing information as negative information and report an error in the structural concept, although the engineer has not finished for example a system, yet.

Example 9 (continued) Suppose that DistributedLoad, Slab, StructuralSystem are atomic concepts and that loads and hasPart are atomic roles. The concept $\mathsf{Slab} \sqcap \neg(\mathsf{DistributedLoad} \sqcup \mathsf{StructuralSystem})$ describes then a slab that is not a load or structural system . If one wants to describe the concept of a distributed load that only loads slabs, one can do this by the expression $\mathsf{DistributedLoad} \sqcap \exists \mathsf{loads}.\mathsf{Slab} \sqcap \forall \mathsf{loads}.\mathsf{Slab}$. The edge between DistributedLoad and Slab for example in figure 3.2 is defined by the former concept expression but misses the quantified role restriction (1,1) that only one distributed load can be applied to one slab. To state this, additional constructs are required. Another example is a structural system that has at least two parts, which can be expressed by $\mathsf{StructuralSystem} \sqcap (\exists_{\leq 2}\, \mathsf{hasPart})$.

The description logic that provides the above introduced constructs for full negation, intersection, number restrictions, union, value restriction, and existential quantification is named $\mathcal{ALCN}$ and is an extension of the well-known description logic $\mathcal{ALC}$ by number restrictions [SSS91]. This description logic can be further extended by other constructs. The resulting group of description logics is termed *expressive* since the logics originate from relationships with expressive modal logics and represent additional information about the relational structure of the domain. Additional constructs of *qualified number restrictions*, *inverse* roles, and *concrete domain predicates* are necessary to represent the domain of multi-storey concrete building structures.

Example 10 (Required constructs for conceptual domain) Nearly-independent systems allow the experienced practitioner to structure the overall design task into controllable subtasks. To define for example a structure with a independent stability and vertical system by the expression

$$\exists_{=1}\mathsf{hasPart}.\,\mathsf{StabilitySystem} \;\sqcap\; \exists_{=1}\mathsf{hasPart}.\,\mathsf{VerticalSystem},$$

qualified number restrictions are required where the independence is described by $\mathsf{StabilitySystem} \sqcap \neg\exists \mathsf{hasPart}.(\mathsf{Column} \sqcup \mathsf{Beam})$ to describe an independent stability system that has no columns or beams as parts. The extension by qualified number restrictions is denoted $\mathcal{Q}$ in the language description.

Consider the concept of a shear wall which is part of a lateral stability system. An inverse role $\mathsf{hasPart}^-$ for the expression $\exists \mathsf{hasPart}^-.\mathsf{StabilitySystem}$ is needed. Furthermore, to express that an element loads another a support relation $\mathsf{supports} = \mathsf{loads}^-$ with its inverse counterpart is needed. It is needed for the concept expression $\mathsf{Column} \sqcap \exists \mathsf{supports}.\mathsf{Beam}$ stating that the element is a column and supports a beam. The language description including this construct is extended by $\mathcal{I}$. This extension gives more inferences, since additional information about a relation is represented.

To express the concept of a column, which is a structural element that must be loaded by a compression force and has to possess certain element dimensions (its height must be smaller or equal four times its breadth, and its length must be at least six times larger than the height), two other constructs for *attributes* and *concrete domain predicates*, denoted $\mathcal{D}$ in the language description, as for example $\mathsf{Normalforce}_{kN}$ and $\leq_0$, are required.

$$\mathsf{StructuralElement} \sqcap \exists \mathsf{Normalforce}_{kN}.\leq_0 \sqcap$$
$$\exists \mathsf{Height}_m, \mathsf{Breadth}_m.\leq_{4\cdot\mathsf{Breadth}_m} \sqcap \exists \mathsf{Length}_m, \mathsf{Height}_m.\geq_{6\cdot\mathsf{Height}_m}$$

I will also model load magnitudes, structural element parameters and state variables, and geometrical constraints by attributes over a concrete domain.

Definition 7 (Concrete domain) *The* concrete domain $\mathcal{D} = \langle \mathbb{R}, \Phi_\mathcal{D} \rangle$ *is a pair, where* $\mathbb{R}$ *is the set of rational numbers, and* $\Phi_\mathcal{D}$ *is a set of predicate names from* L *over* $\mathbb{R}$*. Each predicate* P *from* $\Phi_\mathcal{D}$ *is associated with an arity* n *and a* n*-ary predicate* $P^\mathcal{D} \subseteq \mathbb{R}^n_\mathcal{D}$*. The predicates of* $\Phi_\mathcal{D}$ *are:*

- *unary predicate* $\top_\mathcal{D} = \mathbb{R}$ *and a unary predicate* $\bot_\mathcal{D} = \emptyset$ *for which* $\bot_\mathcal{D}$ *is the negation of the predicate* $\top_\mathcal{D}$*,*
- *unary predicates* P_r *for each* $P \in \{\leq, \geq, =, <, >\}$ *and each* $r \in \mathbb{R}$ *with* $(P^\mathbb{R}_r) = \{r' \in \mathbb{R} \mid r'Pr\}$*, and*
- *binary predicates* P_r *for each* $P \in \{\leq, \geq, =, <, >\}$ *and each* r *a polynomial over* n *real attributes with* $(P^\mathbb{R}_r) = \{r' \in \mathbb{R}^n \mid r'Pr\}$*.*

This concrete domain is admissible according to [BH91] because it is closed under negation and the satisfiability problem for real arithmetic is decidable. The resulting expressive description logic is called $\mathcal{ALCQI}(\mathcal{D})^-$ because it allows no attribute chains. It extends the logic $\mathcal{SHIQ}$ [HST00] by concrete domain attributes but leaves out the constructs for role hierarchies and transitive roles included in $\mathcal{ALCQHIR}^+(\mathcal{D})^-$ [HM01]. Therefore, it is a decidable subset according to [Lut02].

A couple of basic inferences on concept expressions in a description logic exists. *Subsumption* is the task of checking whether a concept D is more general than a concept C, denoted $C \sqsubseteq D$. To check the relation between the two concepts, one has to take the concept expressions in the terminology into account. Another reasoning task is to check the *satisfiability* of a certain concept expression, which amounts to check if the concept expression does not denote the empty concept.

To build up a description logic knowledge base the additional concept of components is necessary. A terminological component contains conceptual knowledge in a *TBox*. It is built up through *declarations* that describe general properties of concepts by concept expressions. Conceptual knowledge can be separated into so-called *definitorial* and *background knowledge* [Sat03]. Definitorial knowledge defines the domain taxonomy - elements or systems with their properties like attributes or aggregations - and background knowledge. It reduces the set of correct concepts. For examples, see also EISFELD and SCHERER [ES02a]. An additional component, called *CBox*, is needed to represent the conceptual knowledge about the load transfer among elements. The ABox contains extensional knowledge in form of facts about individuals of the domain of discourse, here the description of the structural concept under development. Intensional knowledge, stored in the TBox and CBox, is usually thought not to change, whereas extensional knowledge is problem-specific, and therefore subject to constant change while problem solving. A TBox defines usually a partial order among concepts, which is entailed by the subsumption relation. A CBox allows to formulate queries over individuals of the ABox that are known or provable to belong to two concepts.

An ordinary TBox comprises a set of concept declarations. The basic form of a declaration is a *concept definition*, where the new concept is defined in terms of previously defined or atomic concepts. For example, $C \equiv D \sqcap E$ defines the concept C in terms of the atomic concepts D and E. If the requirement of acyclic and atomic concept declarations is dropped, a declaration can also be an *inclusion axiom* of the form $C \sqsubseteq D$. This extension increases the complexity of the inference problems, which even happens for simple cyclic concept definitions shown by SCHILD [Sch91].

Example 11 (Concept definition and axiom of elements) A slab is a structural element that is loaded by a distributed load can be defined by the following declaration:

$$\mathsf{Slab} \equiv \mathsf{StructuralElement} \sqcap \exists \mathsf{loads}^{-}.\mathsf{DistributedLoad},$$

which is interpreted as logical equivalence and represents definitorial knowledge. However, in the conceptual design domain, inclusion axioms are also

needed interpreted as logical implication to represent background knowledge. For example, if an element is loaded by a distributed load then it must be a slab which is represented by

$$\mathsf{DistributedLoad} \sqcap \exists \mathsf{loads}.\mathsf{StructuralElement} \sqsubseteq \mathsf{Slab}.$$

The basic reasoning service for a TBox is *classification*. It computes the concept hierarchy by placing concepts in the proper place of the hierarchy. Classification can be realized by verifying the subsumption relation between neighboring concepts. I employ the classification service for the development of a correct taxonomy of the conceptual design domain.

A CBox is a set of concept definitions not occurring in the TBox. They are used to retrieve elements of an ABox known to be instances of these concepts. The retrieved elements do not refer to the objects of the domain, but what the knowledge base knows about the domain.

Example 12 (Concepts for load transfer) Consider a state in the design process, in which the engineer would like to check, if a column transfers the applied load by a beam to a supporting element and if the column is approximately designed. This requirement can be defined by the concepts:

$$\mathsf{LoadedColumn} \equiv \mathsf{Column} \sqcap \exists \mathsf{loads}^{-}.\mathsf{Beam},$$

$$\mathsf{DesignedColumn} \equiv \mathsf{Column} \sqcap \mathsf{Designable} \sqcap \exists \mathsf{supports}.(\mathsf{Column} \sqcup \mathsf{Beam}).$$

If the element belongs to both concepts, the load transfer is complete. Otherwise, the element is not load-bearing or is not completely designed.

An ABox contains the problem-specific knowledge, which is the structural concept in the design domain. Usually, the structural concept is built up by assertions about individual elements, loads, etc., called *concept assertions*, and arranging elements and systems by *role assertions*. *Predicate* and *attribute assertions* allow to represent element parameters and concrete domain constraints. As a limitation, concept and role expressions can be restricted to denote atomic or defined expressions as for the closed taxonomy assumption.

Example 13 (Representation of the structural concept) Consider a design situation in which an incomplete structural concept is described by having a slab system on which a distributed load is applied. The role assertions $\mathsf{hasPart}(\mathsf{S}, \mathsf{SL})$ and $\mathsf{loads}(\mathsf{LO}, \mathsf{SL})$ state for example that the structure possesses as part a slab system, which supports a distributed load.

The basic reasoning tasks for an ABox are *instance checking* and *consistency testing* that amount to check whether a given individual is an instance of a specified concept and whether every concept in the knowledge base admits at least one individual, respectively. Additionally, I define *completeness testing* as a reasoning task for individuals of the ABox on a CBox. It amounts to checking if an individual is an instance of the difference set of the two defined concepts. If it can be proven that individuals of the ABox are instances of the first but not of the second set the load transfer is incomplete and additional individuals in the ABox representing elements are required. I employ it for finding elements with incomplete load paths in the structural concept. Instance checking and other reasoning services for ABoxes can be reduced to the consistency test problem. Reasoning with respect to a given ABox makes reasoning more complex. Therefore, the techniques for TBox reasoning have to be adjusted. I employ consistency and completeness testing to assist the engineer in constructing a correct structural concept. The completeness test is performed for all individual elements in the ABox.

3.3.2 Language Definition of Description Logic

In the following I define the language and the used reasoning services formally. Concept descriptions allow to build up complex concept expressions about structural object properties in the conceptual design domain.

Definition 8 (Syntax of concepts) *Let* C, R *and* F *be disjoint sets of* concept, role *and* attribute names *from* L *and* $n \in \mathbb{N}$. *In concept expressions,* inverse roles, *denoted* R^-, *maybe used instead of role names* R. *If* C *and* D *are concepts of* C, R *is a role of* R, $P \in \Phi_{\mathcal{D}}$ *is a predicate of arity* n *and* $f_1, \ldots, f_n$ *are attributes from* F, *then concepts can be formed according to the following syntax rules:*

$C, D \longrightarrow$	$\top \mid$	(universal concept or true)
	$\bot \mid$	(bottom concept or false)
	$\neg C \mid$	(full negation)
	$C \sqcap D \mid$	(intersection)
	$C \sqcup D \mid$	(union)
	$\forall R.C \mid$	(value restriction)
	$\exists R.C \mid$	(full existential quantification)
	$\exists_{\geq n} R.C \mid$	(qualified number restrictions)
	$\exists_{\leq n} R.C \mid$	
	$\exists_{=n} R.C \mid$	
	$\exists f_1, \ldots, f_n.P \mid$	(existential predicate restriction)
	$\forall f_1, .\bot_{\mathcal{D}}$	(limited predicate value restriction).

The universal concept $\top$ abbreviates $C \sqcup \neg C$ and the bottom concept $\bot$ the concept $\neg\top$, respectively. To define a formal semantics of $\mathcal{ALCQI}(\mathcal{D})^-$-concepts, an interpretation has to be introduced. It allows to reason about the correctness of intensionally defined structural objects. An interpretation consists of the universe of discourse and an interpretation function that assigns to every atomic concept a subset of the interpretation domain and to every role a subset of binary relations over the interpretation domain. This type of semantics is called Tarski-style semantics for concepts, since it is set-theoretic.

Definition 9 (Semantics of concepts) $\mathcal{I} = \langle \Delta_{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ *is an* interpretation *that consists of a set* $\Delta_{\mathcal{I}}$, *the* abstract domain, *with* $\Delta_{\mathcal{I}} \cap \mathbb{R} = \emptyset$, *and an* interpretation function $\cdot^{\mathcal{I}}$. *The interpretation function maps each atomic*

concept name C to a set $C^{\mathcal{I}} \subseteq \Delta_{\mathcal{I}}$, each role name R to a binary relation $R^{\mathcal{I}} \subseteq \Delta_{\mathcal{I}} \times \Delta_{\mathcal{I}}$, and each attribute name f to a partial function $f^{\mathcal{I}} \colon \Delta_{\mathcal{I}} \to \mathbb{R}$, which is written in the extensional form $(a^{\mathcal{I}}, x^{\mathcal{I}}) \in f^{\mathcal{I}}$. Inverse roles are interpreted as $(R^{-})^{\mathcal{I}} = \{(a,b) \in \Delta_{\mathcal{I}} \times \Delta_{\mathcal{I}} \,|\, (b,a) \in R^{\mathcal{I}}\}$. The interpretation function is then extended to concepts by the following inductive definitions with $n \in \mathbb{N}$:

$$
\begin{aligned}
\top^{\mathcal{I}} &= \Delta_{\mathcal{I}}, \\
\bot^{\mathcal{I}} &= \emptyset, \\
(\neg C)^{\mathcal{I}} &= \Delta_{\mathcal{I}} \setminus C^{\mathcal{I}}, \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}}, \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}}, \\
(\forall R.C)^{\mathcal{I}} &= \{a \in \Delta_{\mathcal{I}} \mid \forall b \in \Delta_{\mathcal{I}} : (a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}, \\
(\exists R.C)^{\mathcal{I}} &= \{a \in \Delta_{\mathcal{I}} \mid \exists b \in \Delta_{\mathcal{I}} : (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}, \\
(\exists_{\geq n} R.C)^{\mathcal{I}} &= \{a \in \Delta_{\mathcal{I}} \mid |\{b \in \Delta_{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}| \geq n\}, \\
(\exists_{\leq n} R.C)^{\mathcal{I}} &= \{a \in \Delta_{\mathcal{I}} \mid |\{b \in \Delta_{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}| \leq n\}, \\
(\exists_{=n} R.C)^{\mathcal{I}} &= \{a \in \Delta_{\mathcal{I}} \mid |\{b \in \Delta_{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}| = n\}, \\
(\exists f_1, \ldots, f_n.P)^{\mathcal{I}} &= \{a \in \Delta_{\mathcal{I}} \mid \exists x_1, \ldots, x_n \in \mathbb{R} : \\
&\qquad (a,x_1) \in f_1^{\mathcal{I}} \wedge \cdots \wedge (a,x_n) \in f_n^{\mathcal{I}} \wedge (x_1, \ldots x_n) \in P^{\mathcal{D}}\}, \\
(\forall f.\bot_{\mathcal{D}})^{\mathcal{I}} &= \{a \in \Delta_{\mathcal{I}} \mid \neg \exists x_1 \in \mathbb{R} : (a,x_1) \in f^{\mathcal{I}}\},
\end{aligned}
$$

where $|\cdot|$ denotes the cardinality of a set. Two concepts are equivalent*, written $C \equiv D$ if $C^{\mathcal{I}} = D^{\mathcal{I}}$ for all $\mathcal{I}$. A concept is called* satisfiable *if there exists an interpretation $\mathcal{I}$ such that $C^{\mathcal{I}} \neq \emptyset$. Such an interpretation is called a* model *of C. A concept C is* subsumed *by a concept D, written $C \sqsubseteq D$, if $C^{\mathcal{I}} \sqsubseteq D^{\mathcal{I}}$ for all interpretations $\mathcal{I}$.*

An interpretation $\mathcal{I}$ corresponds to an intermediate design state. However, the domain $\Delta_{\mathcal{I}}$ is structured by facts stated in the description logic and not by extensionally defining the relations. To define concept abbreviations, previously called declarations, terminological axioms are used. A set of them specifies the conceptual knowledge about DIN 1045-1 constraints and suitable element aggregations for systems in a TBox.

Definition 10 (Syntax and semantics of TBox) *A* terminological axiom *is a* concept definition *$C \equiv D$, a* concept specialization *$C \sqsubseteq D$, where C is a concept name of C and D is a concept term, or a* generalized inclusion axiom*, for which C and D are concept terms. A finite set of such axioms*

is called a TBox $\mathcal{T}$. *Standard semantics is used for* $\mathcal{T}$. *An interpretation* $\mathcal{I}$ satisfies *an axiom of the form* $C \equiv D$ *and* $C \sqsubseteq D$ *if and only if* $C^{\mathcal{I}} = D^{\mathcal{I}}$ *and* $C^{\mathcal{I}} \sqsubseteq D^{\mathcal{I}}$, *respectively. An interpretation* $\mathcal{I}$ *is a* model *for* $\mathcal{T}$ *if and only if it satisfies all declarations in* $\mathcal{T}$. *Two axioms are* equivalent *if they admit the same models.*

When I modeled the conceptual design domain, I constructed a terminology about the conceptual knowledge. During this process, it is important to find out if newly introduced concepts make sense in terms of already defined concepts or if they are contradictory to them. I used the reasoning service of classification to do this.

Definition 11 (Classification) *A concept* C *is* subsumed *by a concept* D *with respect to a TBox* $\mathcal{T}$ *if* $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ *for every model* $\mathcal{I}$ *of* $\mathcal{T}$, *written* $C \sqsubseteq_{\mathcal{T}} D$. *A* subsumption hierarchy *is a quasi ordering over all concept names occurring in* $\mathcal{T}$ *with respect to the subsumption relation* $\sqsubseteq_{\mathcal{T}}$.

A set of concept definitions is used to define the conceptual knowledge about valid load paths for structural elements in a CBox. The CBox represents thus the required structure for a goal concept by separate concept definition for structural element types and their required support relations. They determine if the load transfer for all elements of a developing structural concept is complete where for each subset of structural element types two completion axioms are used to describe their local load transfer.

Definition 12 (Syntax and semantics of CBox) *Let C,D be two concept names of* C, *which do not occur in* $\mathcal{T}$ *and both include an atomic concept name E. E denotes an equivalence class of structural elements. Furthermore, let* $\mathcal{A}$ *be an ABox. A* completion axiom *is a* concept definition $C \equiv D$. *For some abtract object names* $a_1, \ldots, a_k$ *in* $\mathcal{A}$ *belonging to E, there exist two completion axioms. A finite set of such pairs of completion axioms is called a* CBox $\mathcal{C}$ *and C a* defined concept, *respectively. An interpretation satisfies such a pair of completion axioms* C_1, C_2 *for some abstract object names* $a_1, \ldots, a_k$ *for E if* $C_1^{\mathcal{I}} \setminus C_2^{\mathcal{I}} = \emptyset$ *for* $E^{\mathcal{I}} = \{a_1^{\mathcal{I}}, \ldots, a_k^{\mathcal{I}}\} \subseteq \Delta_{\mathcal{I}}$. *Such an interpretation is a called a* model *of a CBox* $\mathcal{C}$ *if it satisfies for every concept* E *in* $\mathcal{C}$ *with given* $a_1, \ldots, a_k$ *the pair of completion axioms.*

I use an ABox for the description of a design state, which represents the structural concept under development. The structural concept description is built up by a set of concept, role, attribute, and predicate assertions.

Definition 13 (Syntax of assertions in ABox) *Let* $\mathsf{O}_\mathcal{D}$ *and* $\mathsf{O}_\mathcal{A}$ *be disjoint sets of so-called* concrete *and* abstract object names *from* L*, abbreviated* $\mathsf{O}_\mathcal{S}$*. If* C *is a concept name of* C*,* R *is a role name of* R*,* f *is an attribute name of* F*,* $P \in \Phi_\mathcal{D}$*,* a *and* b *are elements of* $\mathsf{O}_\mathcal{A}$*, and* $x, x_1, \ldots, x_n$ *are elements of* $\mathsf{O}_\mathcal{D}$*, then the following expressions are an* assertional axiom:

$$\begin{aligned} C(a) \quad & \text{(concept assertion)} \\ \neg C(a) \quad & \text{(negated concept assertion)} \\ R(a,b) \quad & \text{(role assertion)} \\ f(a,x) \quad & \text{(attribute assertion)} \\ P(x_1,\ldots,x_n) \quad & \text{(predicate assertion).} \end{aligned}$$

We call a finite set of such axioms an ABox $\mathcal{A}$*, which represents the concept under development in a design state.*

A semantics to an ABox is given by extending an interpretation to object names that refer to structural objects in the conceptual design domain like elements, systems, etc.. It is assumed that distinct abstract object names denote distinct abstract objects. Therefore, the mapping has to respect the *unique name assumption* (UNA), that is, if a, b are distinct names, then $a^\mathcal{I} \neq b^\mathcal{I}$. The UNA does not hold for concrete object names.

Definition 14 (Semantics of assertions in ABox) *An* extended interpretation *for an assertional language is an interpretation* $\mathcal{I}$*, which, in addition, assigns to every abstract object name* $a \in \mathsf{O}_\mathcal{A}$ *an element* $a^\mathcal{I} \in \Delta_\mathcal{I}$ *and to every concrete object name* $x \in \mathsf{O}_\mathcal{D}$ *an element of* $x^\mathcal{I} \in \mathbb{R}$*. An extended interpretation* $\mathcal{I}$ satisfies *an assertion*

$$\begin{aligned} C(a) \quad & \textit{iff} \quad a^\mathcal{I} \in C^\mathcal{I}, \\ \neg C(a) \quad & \textit{iff} \quad a^\mathcal{I} \notin C^\mathcal{I}, \\ R(a,b) \quad & \textit{iff} \quad (a^\mathcal{I}, b^\mathcal{I}) \in R^\mathcal{I}, \\ f(a,x) \quad & \textit{iff} \quad (a^\mathcal{I}, x^\mathcal{I}) \in f^\mathcal{I}, \\ P(x_1,\ldots,x_n) \quad & \textit{iff} \quad (x_1^\mathcal{I},\ldots,x_n^\mathcal{I}) \in P^\mathcal{D}. \end{aligned}$$

If $a^\mathcal{I} \in C^\mathcal{I}$*, then* a *is called an* instance *of* C *in* $\mathcal{I}$*. If* $(a^\mathcal{I}, b^\mathcal{I}) \in R^\mathcal{I}$*, then* b *is an* R-successor *of* $a^\mathcal{I}$*. We define* $Inv(R)$ *to be* R^- *for a role name* R *and* R *for an inverse role* R^-*. If* b *is an* R*-successor of* a *or* a *is an* $Inv(R)^-$*-successor of* b*, then* b *is called an* R-neighbor *of* a*.* $\mathcal{I}$ *is a design state and a* model *of* $\mathcal{A}$*, if it satisfies all assertions in* $\mathcal{A}$*.*

Note, that we can construct different models and reason about incomplete structural concepts due to open-world semantics if they are represented by an ABox. I use the reasoning service of consistency testing for detecting inconsistent structural concepts or parts of them on the prescribed TBox.

Definition 15 (Consistency testing) *A structural concept represented by an ABox $\mathcal{A}$ is* consistent *with respect to a TBox $\mathcal{T}$ if and only if $\mathcal{A}$ and $\mathcal{T}$ together have a model $\mathcal{I}$, written $\mathcal{I} \models \mathcal{A}, \mathcal{T}$. Otherwise, a structural concept as ABox $\mathcal{A}$ is called* inconsistent.

To find the elements of a structural concept that do not transfer the loads to adjacent elements beneath, I employ the reasoning service of completeness testing based on a combined instance retrieval test.

Definition 16 (Completeness testing) *A structural concept represented by an ABox $\mathcal{A}$ is* complete *with respect to a CBox $\mathcal{C}$ if and only if $\mathcal{A}$ and $\mathcal{C}$ together have the same model, written $\mathcal{I} \models \mathcal{A}, \mathcal{C}$. Otherwise, a structural concept is called* incomplete.

A complete and consistent ABox has a model $\mathcal{I}$ that has the same structure as the goal concept $\mathcal{I}_G$ in definition 2. Examples of testing the consistency and completeness will be given in Chapter 4 in which a detailed example will be explained. According to [BCM+02, DLN+93] the reasoning services of classification of a TBox $\mathcal{T}$, completeness testing of a $\langle \mathcal{A}, \mathcal{C} \rangle$ knowledge base, and consistency testing of a $\langle \mathcal{A}, \mathcal{T} \rangle$ knowledge base can be reduced to the problem of consistency testing of an ABox with an empty TBox. The empty TBox is obtained by a preprocessing step in the algorithm. The same preprocessing can be done for a CBox. A sound and complete tableau algorithm for consistency testing of the description logic $\mathcal{ALCN}\mathcal{H}_{R^+}(\mathcal{D})^-$ is given in [HMW00] that terminates on any input. In the following an algorithm to decide the consistency of $\mathcal{ALCQI}(\mathcal{D})^-$ knowledge base $\langle \mathcal{A}, \mathcal{T} \rangle$ is summarized, which was first developed by [HST00] and adjusted to include concrete domain predicates by [HMW00]. As an example, this algorithm includes the preprocessing step for a TBox. The problem of consistency testing should remain decidable and sound and complete according to the results obtained by [BLSW02] about fusion of different description logics. It is beyond the scope of the thesis to show these properties but the aim to gain a basic understanding how such an algorithm works to find a structural goal concept.

3.3.3 Consistency Algorithm

Tableau algorithms are often used to implement the reasoning service of ABox consistency [BS01]. They originate from FOL tableau calculus where the rules have been adapted to the specific requirements of description logics. Disjunction and qualified number restrictions in the language cause non-determinism, which must be implemented with search techniques. Therefore, the algorithm works with sets $\mathcal{M}$ of ABoxes. $\mathcal{M}_{i+1}$ is obtained from $\mathcal{M}_i$ by an application of a completion rule. If $\mathcal{A}_0$ shall be tested for consistency we start with the singleton set $\mathcal{M}_1 = \{\mathcal{A}_1\}$. Intuitively, the algorithm searches for an ABox in a tree, which does not include an obvious contradiction and to which no more rules can be applied, a so-called *open* and *complete* ABox. Search takes place over sets of ABoxes by applying so-called *completion rules* to assertional axioms of an ABox, which generate new search states. Because more than one rule might be applicable a *completion strategy* is needed. The completion strategy enforces some kind of breadth-first search. The algorithm applies completion rules until no more rules are applicable to the - by assertional axioms augmented - ABoxes in $\mathcal{M}$. Thus, implicit knowledge in the initial ABox is made explicit. The application of rules stops and backtracks to remaining choice points, if an obvious contradiction, a so-called *clash*, is discovered. Therefore, an ABox with an obvious contradiction is called *closed*. When the algorithm from figure 3.3 succeeds to construct an augmented ABox to which no more rules can be applied and which does not contain a clash, then a *completion* of $\mathcal{A}_0$ exists and the ABox is consistent, so that a model for it can be constructed.

define procedure ABox-consistency($\mathcal{A}_0, \mathcal{T}$)
  $\mathcal{A}'_0 :=$ *fork-elimination*($\mathcal{A}_0$)
  $\mathcal{A}_1 :=$ *calculate-augmented*($\mathcal{A}'_0, \mathcal{T}$)
  $r := 1$
  $M_1 := \{\mathcal{A}_1\}$
  while a rule from the *completion rules* is applicable to M_r **do**
    $r := r + 1$
    $M_r :=$ apply-completion-rule($\mathcal{M}_{r-1}$)
    od
  if there is an $\mathcal{A} \in M_r$ that does not contain a *clash* **then**
    return consistent
    return inconsistent

Figure 3.3: ABox-consistency algorithm

To test the initial ABox $\mathcal{A}_0$ for consistency, the algorithm starts with transforming $\mathcal{A}_0$ into an *augmented* ABox $\mathcal{A}_1$ that is consistent with respect to an empty TBox. To obtain $\mathcal{A}_1$, first all so-called *forks* are eliminated. A fork in $\mathcal{A}_0$ is an attribute f with $\{f(a, x_1), f(a, x_2)\} \subseteq \mathcal{A}_0$. Since f is interpreted as a partial function, such a fork means that x_1, x_2 have to be interpreted as the same objects. The *fork elimination* replaces every occurrence of x_2 in $\mathcal{A}_0$ by x_1, denoted *fork-elimination*$(\mathcal{A}_0)$, which returns $\mathcal{A}'_0$ without forks.

After the fork elimination, the algorithm calculates the augmented ABox $\mathcal{A}_1$, denoted *calculated-augmented*$(\mathcal{A}'_0, \mathcal{T})$. Every concept term C occurring in $\mathcal{A}'_0$ is replaced by its definition or specialization. Generalized inclusion axioms $C \sqsubseteq D$ are then replaced by so-called *universal concept assertions*, after rewriting concept definitions into the equivalent form $C \sqsubseteq D$ and $D \sqsubseteq C$. The assertion $\forall x : (\neg C \sqcup D(x))$ are added to the ABox for every generalized inclusion axiom, where x have to be fresh individual names. As a final preprocessing step, every concept term in $\mathcal{A}'_0$ has to be transformed into its negated normal form by *rewriting rules* based on DeMorgan laws and rules for quantifiers.

Definition 17 (Rewriting rules for negation normal form) *A concept term C is in* negation normal form*, denoted $\sim C$, if and only if negation signs occur only in front of concept names from* C*. The negated normal form is obtained by the following* rewriting rules*:*

$$
\begin{aligned}
\neg\neg C &\longrightarrow C \\
\neg(C \sqcap D) &\longrightarrow \neg C \sqcup \neg D \\
\neg(C \sqcup D) &\longrightarrow \neg C \sqcap \neg D \\
\neg\forall R.C &\longrightarrow \exists R.\neg C \\
\neg\exists R.C &\longrightarrow \forall R.\neg C \\
\neg\exists_{\leq n} R.C &\longrightarrow \exists_{\geq n+1} R.C \\
\neg\exists_{\geq n} R.C &\longrightarrow \exists_{\leq n-1} R.C, n > 0 \\
\neg\exists_{\geq 0} R.C &\longrightarrow \bot \\
\neg\exists f_1, \ldots, f_n.P &\longrightarrow \exists f_1, \ldots, f_n.\neg P \sqcup \forall f_1.\bot_{\mathcal{D}} \sqcup \ldots \sqcup \forall f_n.\bot_{\mathcal{D}} \\
\neg\forall f.\bot_{\mathcal{D}} &\longrightarrow \exists f.\top_{\mathcal{D}}
\end{aligned}
$$

If no rule is applicable the resulting concept is in its negation normal form and $\sim C$ and C have the same models.

The rules push negations inside by their exhaustive application. The obtained concept has the same models, since the rewriting rules preserve the equiva-

lence. As a last preprocessing step, inequality assertions $a \not\equiv b$ are added for the abstract individuals contained in the ABox.

After the augmented ABox $\mathcal{A}_1$ has been computed, the algorithm starts with $M_1 := \{\mathcal{A}_1\}$. In accordance with a completion strategy, the completion rules are applied to M_1 as long as possible to generate a complete set of ABox M_r to which no more rules can applied. An *individual ordering* $\prec_I$ and *concrete ordering* $\prec_C$ for new concrete individuals in $\mathcal{A}$ are necessary because first all rules have to be applied to individuals already present in $\mathcal{A}$ before new individuals are introduced.

Definition 18 (Ordering) *If b is an abstract individual newly introduced in $\mathcal{A}$, then $a \prec_I b$ is an* abstract ordering *for all new individuals a already existing in $\mathcal{A}$. If y is a concrete individual new in $\mathcal{A}$, then $x \prec_C y$ is an* concrete ordering *for all concrete individuals x already in $\mathcal{A}$.*

A new individual a is called *blocking* a new individual b from $\mathcal{A} \in \mathcal{M}$, if $\sigma(\mathcal{A}, a) =: \{C \,|\, C(a) \in \mathcal{A}\} \supseteq \sigma(\mathcal{A}, b)$ and $a \prec_I b$. An individual a is called *directly blocking* an individual b from $\mathcal{A}$, if $\sigma(\mathcal{A}, a) = \sigma(\mathcal{A}, b)$ and $a \prec_I b$. Furthermore, an individual a generated by a rule is denoted $new(a)$.

Definition 19 (Completion rules) *Let $\mathcal{M}$ be a finite set of ABoxes and $\mathcal{A} \in \mathcal{M}_i$ of $\mathcal{M}$. The following set of* completion rules *will replace an ABox $\mathcal{A}$ by one $\mathcal{A}'$ or two ABoxes $\mathcal{A}', \mathcal{A}''$ in $\mathcal{M}$:*

$\rightarrow_\sqcap$-rule:
if *1.* $(C \sqcap D)(a) \in \mathcal{A}$ ***and***
2. $\{C(a), D(a)\} \not\subseteq \mathcal{A}$
then $\mathcal{A}' = \mathcal{A} \cup \{C(a), D(a)\}$

$\rightarrow_\sqcup$-rule:
if *1.* $(C \sqcup D)(a) \in \mathcal{A}$ ***and***
2. $\{C(a), D(a)\} \cap \mathcal{A} = \emptyset$
then $\mathcal{A}' = \mathcal{A} \cup \{C(a)\}$ ***or*** $\mathcal{A}'' = \mathcal{A} \cup \{D(a)\}$

$\rightarrow_\forall$-rule:
if *1.* $(\forall R.C)(a)) \in \mathcal{A}$ ***and***
2. a has an r-neighbor b with $C(b) \notin \mathcal{A}$
then $\mathcal{A}' = \mathcal{A} \cup \{C(b)\}$

$\rightarrow_{\exists}$***-rule:***
if *1.* $(\exists R.C)(a) \in \mathcal{A}$ ***and***
2. *a has no r-neighbor b with* $C(b) \in \mathcal{A}$ ***and***
3. $new(a) \Rightarrow \neg\exists c : directly\text{-}blocking(a)$
then $\mathcal{A}' = \mathcal{A} \cup \{C(a), R(a,b)\}$ *where b is not used in* $\mathcal{A}$

$\rightarrow_{\exists P}$***-rule:***
if *1.* $(\exists f_1, \ldots, f_n.P)(a) \in \mathcal{A}$ ***and***
2. $\neg\exists x_1, \ldots, x_n : \{f(a_1, x_1), \ldots, f(a_n, x_n)\} \subseteq \mathcal{A}$
then $\mathcal{A}' = \mathcal{A} \cup \{f(a, x_1), \ldots, f(a, x_n), P(x_1, \ldots x_n)\}$
with $x_1, \ldots, x_n$ *are not used in* $\mathcal{A}$

$\rightarrow_{\exists_{\leq}}$***-rule:***
if *1.* $(\exists_{\leq n} R.C)(a) \in \mathcal{A}$ ***and***
2. *a has* $n+1$ *r-neighbors* $b_1, \ldots, b_n$ *with* $\{C(b_1), \ldots, C(b_{n+1})\} \subseteq \mathcal{A}$ ***and***
3. $b_i \neq b_j \notin \mathcal{A}$ *for some* $i, j, 1 \leq i < j \leq n+1$ ***and***
4. $new(a) \Rightarrow \neg\exists c : blocking(a)$
then $\mathcal{A}' = \mathcal{A}[b_i/b_j]$ *by replacing each occurrence of* b_i *by old* b_j
such that $i, j, 1 \leq i < j \leq n+1$ *where* $b_i \neq b_j \notin \mathcal{A}$

$\rightarrow_{\exists_{\geq}}$***-rule:***
if *1.* $(\exists_{\geq n} R.C)(a) \in \mathcal{A}$ ***and***
2. *a has less than r-neighbors* $b_1, \ldots, b_n$ *with* $\{C(b_1), \ldots, C(b_n)\} \subseteq \mathcal{A}$ ***and***
3. $b_i \neq b_j \notin \mathcal{A}$ *for some* $1 \leq i < j \leq n$ ***and***
4. $new(a) \Rightarrow \neg\exists c : blocking(a)$
then $\mathcal{A}' = \mathcal{A} \cup \{R(a, b_i) \mid 1 \leq i \leq n\} \cup \{C(b_i) \mid 1 \leq i \leq n\} \cup$
$\{b_i \neq b_j \mid 1 \leq i < j \leq n\}$ *with n newly created* b_i *and* $b_i \neq b_j \notin \mathcal{A}$

$\rightarrow_{choose}$***-rule:***
if *1.* $\{(\exists_{\leq n} R.C)(a), R(a,b)\} \subseteq \mathcal{A}$ ***and***
2. $C(b)$ *or* $\neg C(b) \notin \mathcal{A}$
then $\mathcal{A}' = \mathcal{A} \cup \{C(b)\}$ ***or*** $\mathcal{A}' = \mathcal{A} \cup \{\neg C(b)\}$

The rules $\rightarrow_{\sqcup}$, $\rightarrow_{\exists_{\leq}}$ *and* $\rightarrow_{choose}$ *are called* non-deterministic *because they can yield different ABoxes* $\mathcal{A}'$ *applied to* $\mathcal{A}$*. The remaining rules are called* deterministic*. Moreover, the rules* $\rightarrow_{\exists}$, $\rightarrow_{\exists P}$ *and* $\rightarrow_{\exists_{\geq}}$ *are* generating *since they can introduce new objects.*

The completion strategy determines according to an ordering of new individuals, which rule is applied if a set of rules is applicable. First, rules are only applied to old individuals before they can be applied to new ones. A rule is applied to a new individual only if no rule is applicable to another individual that has precedence defined by $a \prec_I b$. Completion rules are applied in two steps: first, non-generating completion rules are always applied

as long as possible. If a set of applicable non-generating rules is empty this step is simply skipped. Second, an applicable generating rule is applied and restarts with a non-generating rules. If the set becomes empty, the algorithm restarts with the application of non-generating rules. If a clash is discovered, the algorithm backtracks to remaining choice points or returns inconsistent.

Definition 20 (Clash) *An ABox $\mathcal{A}$ is* closed *if it contains one of the following* clashes*:*

- primitive clash: $\{C(a), \neg C(a)\} \subseteq \mathcal{A}$,
- agreement clash: $\{a \neq a\} \subseteq \mathcal{A}$,
- number restriction clash: $(\exists_{\leq n} R.C)(a) \in \mathcal{A}$ *and* a *has* $n+1$ r*-neighbors* b *with* $b_i \neq b_j$ *for all* $0 \leq i < j \leq n$,
- concrete domain attribute clash: $\{f(a,x), (\forall f.\bot_{\mathcal{D}})(a)\} \subseteq \mathcal{A}$, *and*
- concrete domain predicate clash: $P(x_1^{(1)}, \ldots, x_n^{(1)}) \in \mathcal{A}, \ldots,$ $P(x_1^{(k)}, \ldots, x_{n_k}^{(k)}) \in \mathcal{A}$ *and* $\bigwedge_{i=1}^{k} P_i(x_1^{(i)}, \ldots, x_{n_i}^{(i)})$ *is not satisfiable in* $\mathbb{R}$.

An ABox $\mathcal{A}$ is open *if non of the clashes occurs and called* complete *if no more completion rules are applicable to $\mathcal{A}$. An open and complete ABox $\mathcal{A}$ is a* completion *of $\mathcal{A}_0$.*

An ABox containing a clash is obvious unsatisfiable. If a completion exists, then the ABox is consistent with respect to the TBox and admits a model [BCM+02]. It can be constructed as a canonical interpretation of the completion, which is not described here (for more information see [HMW00]). In Chapter 4, this model construction step will be given for the detailed example.

In the next subsection, I start with giving a brief introduction to planning and actions. The actions work on an ABox, which is the state description of the structural concept under development. The introduced reasoning services of consistency and completeness testing check the correctness of a state description. The description logic with the two reasoning services provides the foundation of the model-based design support to direct the design process.

3.3.4 The Planning Language

Reasoning about actions is a necessary element of intelligent behavior [Wil88]. Planning techniques lent themselves therefore to observed human problem solving, which is goal-directed and differential [NS72]. The task of coming up with a sequence of actions that transforms an initial state[3] into a goal state is called *planning* [RN95]. The reasoning is more specifically called *task planning* because the planning is abstract in nature in comparison to other planning types [GNT03, Rin03]. Todays task planning methods descended from the historical most influential STRIPS[4] planner, which has been extended or modified by appropriate search methods and richer representations of the planning domain. In the STRIPS planning system, world states are represented by sets of first order formulae, and actions change state descriptions by adding and deleting formulae [NG89]. This approach assumes that the planner has complete information about the world - variables are not allowed in state descriptions - and no sensing actions are required to update the state information [Thi03]. Because FOL is very expressive and its processing computational expensive, the language constructs have been restricted for representing the state of the world and the actions. This class of obtained planning languages is propositional and the planning problems possess a lower complexity [Neb00]. However, the relational structure of the domain has to be modeled. Therefore, an ABox $\mathcal{A}$ is used as state description. This ABox represents a set of interpretations and can hence model incomplete structural concepts.

Planning is necessary to reduce the search to relevant actions. Consider for example, an engineer that has to design a structural concept. He would be overwhelmed by the number of actions he can take. Therefore, he has to reason about how design actions taken will affect the developing concept and preclude irrelevant design actions. He can further restrict his focus on those design actions, which achieve a certain task. The ability to *decompose* the design problem into nearly-independent subproblems contributes to the engineer's efficiency, too, because he has to consider less relevant design actions at a time [Yan98]. Thus, the engineer can work on subgoals independently, but needs to do some additional work to combine the resulting subplans for the subtasks. Decomposition requires a second state component besides the concept description to store the current state of decomposition. On this state component planning actions operate. The outcome of the engineer's

[3] A state is similar to a situation of the world if one assumes that actions always succeed and the world is static.

[4] STRIPS stands for STanford Research Institute Problem Solver

planning is usually a set of design actions on the conceptual level, with ordering constraints on them, for execution. Because the engineer has a restricted working memory capacity he has to interleave planning and execution, called *contingency planning* [RN95, Rin03].

Rephrased in a state model, it is the task of a planning system to find a path from the initial state to the goal state and to reduce iterations in the search process. If a goal state, which is fixed by the conceptual knowledge, can be reached by a sequence of actions $[a_1, \ldots, a_n]$ then this sequence is called a *plan* and the search process *state-space* based [GNT03]. In state-space search each node represents a state of the world and a plan is defined as a path through the space. If the planner searches in the space of possible plans, where a plan is defined as a set of actions, plus some constraints over it, the search is called *plan-space* based [Kam97]. This leads also to the distinction of *linear* and *partial-order planners*[5] since the latter allow partial orders of actions during the search process [Wel99].

Depending on the assumptions made about the dynamical system concerning the properties of actions and states representing the world, different types of planning techniques are needed to find an action trajectory from the initial to a goal state. The basic assumptions[6] for state models that led to the so-called *classical planning* are: determinism, full observability, discrete time and actions, finite states, and no control knowledge and underlying domain information [RN95, Wil88, Rin03]. I will discuss these assumptions and state which of them have to be modified to support designing on the introduced model of conceptual design.

In the simplest form, the initial state of the world and the actions that have been taken so far determine the current state of the world, which leads to determinism. This assumption holds for the conceptual design domain because it is static - the engineer is the only an agent triggering events - and the actions themselves are deterministic in nature.

For deterministic planning problems with one initial state there is usually no need to observe the environment because goals can always be reached by one sequence of actions. This is also true for design but an enormous large state space and the engineer's limited working memory capacity require to interleave planning and execution. Interleaving means feedback from the environment for the engineer to guide his design process and set up subgoals in primitive tasks, which is called *closed-loop planning* according to [BG01].

[5] Partial order planners are also called *least-commitment* planners since orderings are only introduced during the search if required [Wel94].

[6] For further reaching assumptions to constrain the state model see [Rin03, GNT03].

The assumption of discrete time and actions enforce unit durations for them. This means that all changes resulting from an action are immediately visible. Since I only consider deliberative problem solving and a static world, this assumption holds for the conceptual design domain, too.

The assumption about finitely many states holds for routine design but is irrelevant since the design state space is very large. This assumption was introduced in the beginning of planning research to retain decidability of the considered planning problems. It restricts conditions in the actions to be function-free literals. Because the engineer generates new design objects, this requirement does not hold for the conceptual design domain. Therefore, decidability cannot be ensured any longer by the system. Since the system shall interactively support the engineer in the decomposition of conceptual design into suitable subtasks the engineer has to decide when to stop the design process. However, the system can check if there are remaining subtasks or the structural concept is incorrect.

In a classical planning framework plans are synthesized by actions based on how they affect the world. There might be however other information resources that can control the planning process. In hierarchical planning for example, so-called control knowledge is represented in hierarchical task networks, to provide information about the structure of possible plans [Yan98]. A system state then incorporates not only the world state but also the refinement state of the task network as introduced in Chapter 2. Another extension is to evaluate the current state for directing the search process [NSE98]. In the conceptual structural design domain, the engineer exploits both information sources. First, he draws on the decomposability of the conceptual design task into nearly self-contained subtasks. Second, he evaluates always the current design state in terms of his conceptual knowledge to avoid unnecessary backtracking. Domain information is usually given by a causal theory of the underlying domain [Wil88]. WILKINS states that practical planning requires a richer model than only the planning domain [WD01]. Therefore, a model about the domain objects is often used upon which conditions are defined. The conceptual knowledge in the TBox and CBox represents this model. In this way, additional control information about correct design states can be supplied to guide the search.

Because states have certain meaning and determine, which actions are applicable in these states, describing extensionally system states is not most suitable for stating facts about it [Rin03]. An ABox $\mathcal{A}$ and a list of remaining subtasks $t = [t_1, \ldots, t_n]$ describes theses facts in the formalism. This ABox - combined with completeness and consistency testing - and the list of subtasks can then be used to guide the search for a suitable sequence of

actions during design. An initial concept $\mathcal{I}_0$ is given by a consistent ABox $\mathcal{A}_0$ and successive system states are reached by planning actions on the task level or design actions on the conceptual level. At the end of design, when the list of subtasks on the stack is empty, a model $\mathcal{I}_G$ for such a complete and consistent ABox is constructed.

Design actions A_D are defined in terms of their applicability to a state description and a simple subtask and how they change the state by deleting or adding assertional axioms from the current ABox. Often, there are regularities in the set of design actions because objects of the domain behave in the same way. This leads to the introduction of *design operators* which include a set of variables $\{?\mathsf{x}_1, \ldots, ?\mathsf{x}_n\}$ as placeholders for design object names $\mathsf{O}_\mathcal{S}$. Variables have to be first substituted by names of domain objects before an operator can be applied to a state in form of a design action. A *substitution* is a mapping $\tau = \{?\mathsf{x}_1/\mathsf{o}_1, \ldots, ?\mathsf{x}_n/\mathsf{o}_n\}$ from the set of variables into the set of domain object names. Thereby, a set of similar actions can be expressed by one operator, also called a STRIPS operator. An *extended* substitution is a substitution for which additional variables not occurring in the precondition have been replaced by object names coming from user input. A design operator is a 5-tuple $\langle name, pre, bin, del, add\rangle$. The *name* includes the name of the simple subtask to which the operator can be applied. The *precondition*, *delete list*, and *add list* being sets of assertional axioms in which the names of the individuals are replaced by variables. A *binding* set ensures that all domain variables are instantiated in the add and delete list by user input. The add and delete list are called *effect* of an operator because they determine how the state is updated, resulting in a *successor state*. Note, that calculations can also establish bindings for variables in the binding set. An operator is *ground* if all the variables in the precondition and effect are substituted by names of domain objects already in the ABox or introduced by the binding. A ground design operator is called an design action. Intuitively, an operator is *applicable* to an ABox and a simple subtask if a variable substitution exists such that the ground precondition is a subset of the ABox and the name matches the task name.

Example 14 (Design operator and its application) Consider for example the design operator a_D for a column:

$$\begin{aligned} a_D \;=\; \langle & \mathsf{makeColumn}, \\ & \{\mathsf{Beam}(?\mathsf{x}_1)\}, \\ & \{\mathsf{setColumn}(?\mathsf{x}_2)\}, \\ & \{\}, \{\mathsf{supports}(?\mathsf{x}_2, ?\mathsf{x}_1)\}\rangle. \end{aligned}$$

The operator is applicable to a state $\mathcal{A}_1 = \{\mathsf{Beam(B1)}\}$ and a simple task $t_S = \mathsf{makeColumn}$ because the operator name matches the task name and $\{\mathsf{Beam(B1)}\} \subseteq \mathcal{A}_1$ under $\tau = \{?\mathsf{x}_1/\mathsf{B1}, ?\mathsf{x}_2/\mathsf{C1}\}$ with $\mathsf{C1}$ the new column object as user input. The application of the design action yields a successor state with $\mathcal{A}_2 = \mathcal{A}_1 \cup \{\mathsf{supports}(?\mathsf{x}_2, ?\mathsf{x}_1)\tau\}$. Delete and modify actions are similar but defined over different set operations in the effect.

In classical planning it is not possible to describe how complex tasks have to be performed. The hierarchical organization by task descriptions is an important technique to reduce the complexity of designing by control information [ENS95]. A design domain includes then additionally to the set of design operators a number of task descriptions about how a task is decomposed into a number of subtasks at a more detailed level by planning actions A_P. Not decomposable tasks are considered to be *simple* from which design operators are referenced, whereas *compound* tasks are decomposed by planning actions into sequences of task descriptions at lower level. This kind of task description is called *hierarchical task network*, abbreviated HTN [Yan98], and originates from early work of SACERDOTI about skeleton plans [Sac80]. The lack of theoretical understanding delayed the progress of HTN planning for which theoretical semantics were first introduced by EROL and HENDLER in [EHN94]. EROL also showed that HTN planning is more expressive than classical planning because cycles in plans and interleaving among subtasks may occur as in design [Ero95]. HTN planning employs the refinement planning model for which similar to plan-based search a compound initial task as plan is refined by decomposing it into subtasks through the hierarchy into greater levels of detail. To represent control information about the structure of possible plans on lower abstraction level, the planning language must be extended to include a construct for the hierarchical task network. It refers to a partially ordered set of subtasks $\{t_1 \prec t_2, \ldots, t_k \prec t_l\}$. Thereby, recipes can be specified to perform complex tasks. A *task* specifies an abstract goal network, which has to be realized by subtasks on lower abstraction level, called *task reduction*. Interactions of subtasks are resolved by protection of interfering conditions, called constraints among subtasks.

In partial-order planning it is hard to reason about the states since the states are not totally instantiated during the search. This led to the reduction of full HTN planning to *hierarchical ordered task planning*, developed by NAU *et al.* [NCLMA99, NCLMA01]. In this planning framework, task networks have to be completely ordered resulting in the list form $[t_1, \ldots, t_l]$. It is an adaptation of HTN planning to the needs of practical applications because complex reasoning like logical inference, calls to external programs, etc. have to be performed for the current state while planning to prune large parts of

the search space [NSE98]. This need applies also to conceptual design where the design state space is very large and the number of possible design actions must be reduced. The engineer therefore not only evaluates the incomplete structural concept on the conceptual knowledge but also decomposes conceptual design into suitable subtasks. Whenever he wants to plan the next task he has already planned everything that comes before, to know the current state of the world. Therefore, the network has to be totally ordered, called a *tasklist*, and the subtasks in this extensions have to be decomposed from left to right. Thus, tasks have the same order in planning and later execution, which allows their interleaving, which is also sometimes called *replanning* [Wil88, Mye96]. Another advantage of this planning technique is that the search strategy allows forward planning from the initial state and backward planning from the goal state at the same time since task networks represent the goal structure [NCLMA01].

Because regularities exists in the planning actions *methods* are introduced, similar to operators for actions. They describe how compound tasks are decomposed. These methods represent control knowledge about the design process in form of task decompositions and groupings of design actions into simple subtasks. A method is 3-tuple $\langle name, pre, red \rangle$. The *name* includes the name of the compound task to which the method can be applied. The *precondition* is the same as for a design operator. The *reduction* is a sequence of subtasks $[t_1, \ldots, t_l]$ at a lower task level. The reduction might include simple or compound tasks. A method with a ground precondition is called a planning action.

Example 15 (Method and its application) Consider the focused compound task $t_1 = \mathsf{designVerticalSystem}$ in a system state $\mathcal{S}_1$ with a structural concept description $\mathcal{A} = \{\mathsf{VerticalSystem(VS)}, \mathsf{Concept(C)}, \mathsf{hasPart(C, VS)}\}$. The method

$$\begin{aligned} a_P \;=\; & \langle \mathsf{designVerticalSystem}, \\ & \{\mathsf{hasPart}(?\mathsf{x}_2, ?\mathsf{x}_1)\}, \\ & [\mathsf{designColumnSystem}, \mathsf{designBeamSystem}] \rangle \end{aligned}$$

is applicable to $\mathcal{S}_1 = (\mathcal{A}, [t_1, \ldots, t_n])$ and results as planning action in the successor state

$$\mathcal{S}_2 = (\mathcal{A}, [\mathsf{designColumnSystem}, \mathsf{designBeamSystem}, \ldots, t_n]).$$

For a simple subtask, the number of design actions is hence reduced the engineer can choose from. The system aids him through the design process by offering suitable reductions to the engineer.

Some description logics have been proposed for including planning facilities. For an overview of them see [AF00] and for single extensions (cf. [DINR96, LS02]). They use for example the system CLASSIC to encode a modal operator by rules to model state transitions or enhanced STRIPS style operators with semantics to capture the relational structure of the planning domain. However, these formalisms are only in a limited way employable to the conceptual design domain because they lack constructs for representing control information necessary to represent a conceptual design domain like the one of building structures. In the next subsection, I define a hierarchical planning language. It uses ideas from [LS02] where preconditions, bindings and effects are defined in terms of restricted description logic expressions.

3.3.5 Planning Language Definition

The planning language encodes the actions to change the system state. Control knowledge is encoded in the preconditions and task reductions for compound tasks at the task level and in the preconditions of operators for simple tasks. Testing actions A_T are differently represented. Testing the consistency of the developing structural concept is encoded as a condition in the preconditions of design operators and methods. This ensures that the engineer does not continue his design on an inconsistent concept. The predicate $\mathsf{StateConsistentp}()$ for an ABox makes a call to the reasoning service of consistency testing of a current ABox, denoted $\mathcal{A}, \mathcal{T} \vdash \{\top, \bot\}$ where $\top$ is returned for a consistent and $\bot$ for an inconsistent ABox. If the design state is consistent the engineer can continue. Otherwise he has first to modify the current concept to regain consistency before continuation. Testing the completeness of a structural concept is carried out on demand of the user and at the end of a simple subtask. A testing action makes a call to the reasoning service of completeness testing of a current ABox, denoted $\mathcal{A}, \mathcal{C} \vdash \{\top, \bot\}$. It returns in case of incompleteness $\bot$ and the set of incomplete individuals for a structural system. Otherwise, such a testing action returns $\top$ and removes the head of the tasklist because the task is considered finished. Actions to finish a simple subtask are modeled as testing actions for which a call to the completeness test is done if required. A tasklist describes the actions that remain to be performed in order to develop a correct structural concept.

Definition 21 (Task and tasklist) *Let* T_S *and* T_C *be two disjoint sets of* simple *and* compound task names *from* P. *If* $t_1, \ldots t_n \in \mathsf{T} = \mathsf{T}_S \cup \mathsf{T}_C$ *then* $t = [t_1, \ldots, t_n]$ *is a* tasklist.

A system state consists of an ABox and a tasklist. During design, this state is changed by the algorithm that will be later presented.

Definition 22 (System state) *If $\mathcal{A}$ is an ABox and t is a tasklist a* system state *is a tuple $\mathcal{S} = (\mathcal{A}, t)$. The head of the tasklist is called* focused.

Conditions defined as parameterized assertional axioms specify preconditions and effects, which determine when a design operator or method is applicable and how a design action affects the current design state.

Definition 23 (Condition) *Let O_X be a set of* object variables*, disjoint from $\mathsf{O}_\mathcal{S}$. A* condition *c is an assertional axiom*

$$C(?x_1),\ R(?x_1, ?x_2),\ f(?x_1, ?x_2),\ P(?x_1, \ldots, ?x_n)$$

where the concrete and abstract object names of $\mathsf{O}_\mathcal{S}$ have been replaced by object variables $?x_1, \ldots, ?x_n \in \mathsf{O}_X$. A condition is ground *if all object variables of O_X have been replaced by names of $\mathsf{O}_\mathcal{S}$. A* test condition *is a predicate $\mathsf{StateConsistentp} : \mathcal{A}, \mathcal{T} \vdash \{\top, \bot\}$ for consistency testing of a current structural concept under development.*

Design operators are stored in the knowledge base of the system. They turn into design actions after variable substitution.

Definition 24 (Syntax of design operators) *Let T_S be the set of simple task names. A* design operator *is a 5-tuple $a_D = \langle name, pre, bin, del, add \rangle$ with a* name *$\mathsf{t}_S \in \mathsf{T}_S$, a* precondition *as set pre of conditions including always a test condition, a* binding *as set bin of* new abstract object assignments *$?x \rightarrow a$ with $a \in \mathsf{O}_\mathcal{A}$ or* new concrete object assignments *$?x \rightarrow x$ with $x \in \mathsf{O}_\mathcal{D}$, an* delete list *as set del of conditions, and an* add list *as set add of conditions. A precondition is* ground *if all elements of pre are assertional axioms. An operator is ground if all design variables have been replaced by object names. $A_D = \{a_{D_1}, \ldots, a_{D_n}\}$ is the set of design operators.*

Object assignments are carried out by user input. For concrete objects, an unknown value can be assigned by a calculation. The application of design operators to a state define the successor state by set operations on the current ABox. In the case of a synthesis action, the successor state has to contain additional assertional axioms for the new structural elements and design information. In the case of delete actions, assertional axioms are subtracted from the current ABox. Modifying actions, are performed by combined addition and subtraction of assertional axioms.

Definition 25 (Design operator application) *Let a_D be a design operator and $\mathcal{S}$ a system state. The design operator a_D is* applicable *in a state $\mathcal{S} = (\mathcal{A}, t = [t_S : x])$ if there is a substitution for a_D such that $pre_\tau = \{c_1\tau, \ldots, c_k\tau\} \subseteq \mathcal{A}$ and $name(a_D) = t_S$. If a_D is ground under the extended substitution of $\tau_E \supseteq \tau$ called a* design action*, and $\mathcal{S}$ a system state then the successor state is obtained under the ground add list $add_{\tau_E} = \{c_1\tau_E, \ldots, c_l\tau_E\}$ and delete list $del_{\tau_E} = \{c_1\tau_E, \ldots, c_m\tau_E\}$ as follows:*

$$\delta(\mathcal{A}, t_S, a_D) = \begin{cases} (\mathcal{A} \cup add_{\tau_E}, t_S) & \textit{for a synthesis action,} \\ (\mathcal{A} \setminus del_{\tau_E}, t_S) & \textit{for a delete action,} \\ (\mathcal{A} \setminus del_{\tau_E} \cup add_{\tau_E}, t_S) & \textit{for a modify action.} \end{cases}$$

This operation implements the transition function for design actions. Testing the load transfer for completeness is covered by a different operator since it manipulates the tasklist of a system state.

Definition 26 (Syntax of test operator) *Let T_S be the set of simple task names. A* test operator *is a 3-tuple $a_T = \langle name, pre, com \rangle$ with a* name $\mathsf{t}_S \in \mathsf{T}_S$*, the empty set as* precondition *pre being always* ground*, and a* complete condition *com. The condition com is a predicate* $\mathsf{StateCompletep} : \mathcal{A}, \mathcal{C} \vdash \{\top, \bot\}$*. $A_T = \{a_{T_1}, \ldots, a_{T_n}\}$ is the set of test operators.*

The predicate tests the completeness of a current structural system under development in a focused and simple tasks. The operator removes the focused task from the tasklist in case of completeness.

Definition 27 (Test operator application) *Let a_T be a test operator and $\mathcal{S}$ a system state. A test operator a_T is* applicable *in a system state $\mathcal{S} = (\mathcal{A}, t = [t_S : x])$ if $name(a_T) = t_S$. It is called a* test action *for a simple subtask t_S. The successor state is obtained as follows:*

$$\delta(\mathcal{A}, t_S, a_T) = \begin{cases} (\mathcal{A}, x) & \textit{for } \mathsf{StateCompletep} : (\mathcal{A}, \mathcal{C}) = \top, \\ (\mathcal{A}, t) & \textit{for } \mathsf{StateCompletep} : (\mathcal{A}, \mathcal{C}) = \bot. \end{cases}$$

The system holds out a range of methods that encode also control knowledge to decompose compound tasks by planning actions. Methods are applicable in different states depending on the developing concept whereby suitable task decompositions are chosen by preconditions.

Definition 28 (Syntax of methods) *Let* T_C *be the set of compound task names. A* method $a_P = \langle name, pre, red \rangle$ *is a 3-tuple with a* name $\mathsf{t}_C \in \mathsf{T}_C$*, a* precondition *as set pre of conditions including a test condition, and a* reduction *red as tasklist. A method is* ground *if all elements of pre are assertional axioms.* $A_P = \{a_{P_1}, \ldots, a_{P_n}\}$ *is the set of methods.*

Methods for which their preconditions are satisfied by the current system state are offered for task decomposition. After their application the head of the tasklist is replaced by the reduction in the method's body, which has to be in turn realized. The semantics of a method application is defined in terms of a tasklist reduction.

Definition 29 (Method application) *Let* a_P *be a method and* $\mathcal{S}$ *a design state. The method* a_P *is* applicable *in a state* $\mathcal{S} = (\mathcal{A}, t = [t_C : x])$ *if there is a substitution for* a_P *such that* $pre_\tau = \{c_1\tau, \ldots, c_k\tau)\} \subseteq \mathcal{A}$ *and* $name(a_P) = \mathsf{t}_C$*. Such a ground method is called a* planning action *and the result is obtained by* $\delta(\mathcal{A}, t_C, a_P) = (\mathcal{A}, [red(a_P), x])$*.*

At intermediate task levels the engineer can decompose the tasks until he reaches the conceptual level, where he can select between design actions to extend the ABox. A range of applicable design actions in a current system state is offered for application because the engineer should remain in control of designing the concept by suitable elements. In the next section, the interactive planning algorithm is presented to support design at the task level.

3.3.6 Interactive Planning Algorithm

I start with relating the constituents of the model M_{CD} from definition 4 to the formalism defined. The initial system state given as problem specific input to the algorithm is $\mathcal{S}_0 = (\mathcal{A}_0, t_0)$. The consistent initial ABox $\mathcal{A}_0$ represents the concept and the compound initial task $t_0 = \mathsf{designStructuralConcept}$ the task to be performed. $\mathcal{A}_0$ provides all required design information, which is already known prior the solution process. This is typically design information that can be looked-up from tables like material properties, occupancy classes, safety factors, etc.. The set of system states $\mathcal{S}$ is not completely constructed prior problem solving but only successively by actions. This is necessary because the search space is rather large due to possible attribute assignments of elements, element combinations by composition and support relations, and possible task decompositions. The set of actions are persistent knowledge. They are the operators A_D and A_T and the methods A_P. The set of tasks remains unchanged. The algorithm returns a goal system state $\mathcal{S}_G = (\mathcal{A}_G, [])$

together with a plan $\mathcal{P} = [a_1, \ldots, a_n]$. For a state to be a goal state, the final ABox $\mathcal{A}_G$ must be consistent and complete.

The algorithm decomposes the head t_1 of the tasklist $t = [t_1 : x]$ of the system state $\mathcal{S} = (\mathcal{A}, t)$ into a list of subtasks at a lower task level by planning actions. The head of the list is always first decomposed until a simple task is reached on the conceptual level. On this level, the engineer develops by design actions the structural concept. By test and finish actions the engineer can determine for simple tasks, if they are complete. The algorithm uses depth-first search and stores intermediate design plans in the nodes. For operator as well as method applicability, a breadth search step is included.

```
define procedure CONED(A, t, A_D, A_T, A_P)
        if t = [] then return []
        if t_1 ∈ T_S then
          A_app := {a_D,T ∈ A_D ∪ A_T | pre(a_D,T) is ground and
                    a_D,T is applicable to t_1 in A}
          a_D,T := user-choose(A_app)
          π := CONED(δ(A, t_1, a_D,T), A_D, A_T, A_P)
          if π = failure then
            return failure
            else return a_D,T.π
        else if t_1 ∈ T_C then
              M_app := {a_P ∈ A_P | a_P is ground and
                        a_P is applicable to t_1 in A}
              if M_app = ∅ then
                return failure
                else a_P := user-choose(M_app)
                     π := CONED (δ(A, t_1, a_P), A_D, A_T, A_P)
                     return a_P.π
```

Figure 3.4: Interactive planning algorithm

The algorithm in figure 3.4 starts searching for an applicable operator or method in the knowledge base for head t_1 of the tasklist t. In case, it is a simple task and a set of operators is applicable to it in the current system state, they are displayed for user choice. The engineer can now select an action due to his preference. Either he selects a design action to develop the structural concept or he checks the subtask for completeness by a testing action. If the simple task is complete, it is removed from the tasklist, otherwise it remains on the list since the task is incomplete. If the ABox $\mathcal{A}$ of the successor state, which is obtained by the design action application, remains consistent, the algorithm

is recursively called with the new ABox. The algorithm offers a reduced operator set for an inconsistent ABox because the predicate StateConsistentp makes all operators that are not suitable to remove the parts that caused the inconsistency unavailable.

In case, t_1 is a compound task specified in the tasklist t and a set of methods is applicable in the design state, the methods are displayed for user selection. The engineer can choose a method due to his preference, again. The reduction of the method specified in the method's body is appended in front of the tail x by a method application and the algorithm is recursively called with the expanded tasklist. If the set of applicable methods is empty, the algorithm returns a failure. If the tasklist t becomes empty, the final structural concept $\mathcal{A}_G$ is consistent and complete on the conceptual knowledge. The algorithm returns the structural concept $\mathcal{A}_G$ and additionally the complete action sequence $\mathcal{P}$ stored in π.

In the next chapter, a detailed example of a computer-aided design with the prototype system is given based on the introduced formalism.

Chapter 4

Prototype System

The goal of this chapter is to present an envisaged design session with the prototype system. At the beginning, the motivation for a graphical user interface and its components are explained. The interface components make the design support available based on the underlying formalism. Along the session, the extended design support is explained in terms of the requirement specification. It highlights the different types of support to improve the user's design performance. Finally, possible application scenarios of the prototype system are listed.

4.1 GUI Design of the System

Graphical user interfaces (GUI) of todays computer aided drafting (CAD) systems have to allow for direct manipulation of graphical objects by pointing devices like the mouse in a window-based interface. The advantage of such window-based interfaces is that the user can directly manipulate the objects of the domain model, here the model of the structural concept, which supports intuitive working with a CAD system. As SHNEIDERMAN points out [Shn98] this is especially important for explorative tasks like design. Therefore, the computer-aided conceptual design system developed in this thesis possesses the same main GUI components as todays CAD systems, which are a drawing window for direct object manipulation, a palette of commands to trigger design and testing actions, and a documentation line for system output [Inf97]. The high amount of functionality of modern CAD systems has led to a high number of available commands at their GUIs. This increases not only errors but also lowers the productivity of users, especially new ones.

The prototype system tries to cover this usability requirement by an additional GUI component, which usual GUIs of CAD systems lack. It displays the task of conceptual design with its corresponding subtasks and dependencies in a task structure dialog. The dialog restricts the number of available commands the user can select from. Furthermore, the system offers a reasonable sequence of design commands to reduce the number of redesign steps. The formalism with its reasoning services makes this possible because it computes how tasks are suitably decomposed into subtasks and which design action commands are available under a simple task in the task structure.

To design these main components of the GUI for the prototype system I employed *usability engineering*. Usability engineering provides methods to design user-centered interactive systems. For a list of them see [Shn98] and for a summary of the most prominent ones [Wol03]. Because the GUI design was not the main part of the thesis, I selected the *discount usability engineering* approach, proposed by NIELSEN [Nie90]. The approach focuses on four major techniques. *User* and *task observation* is carried out to get a clear notion of the user's task, in my case conceptual design of concrete building structures. This was covered in Section 2.5 "Design Studies" of the thesis. The resulting task description has to be mapped to *scenarios for testing*. A scenario features a part of or the complete system's user interface, which can be realized by prototypes or "mockups", to test the design with users. I used paper prototypes proposed by RETTIG in [Ret94] that allow a fast and cheap production of scenarios. They can in turn be tested by *simplified thinking aloud* studies in iterative cycles. I conducted three studies with different engineers to identify major usability problems. The engineers had different levels of experience to cover a range of potential system users in the future. NIELSEN mentions in [Nie94] that three to five users are often sufficient to determine 75 per cent of the usability problems. The final step, a *heuristic evaluation* of the prototype is conducted by an usability expert following general evaluation rules or guidelines, like for CAD systems given in [Inf97]. This step has not been carried out, yet, since the whole GUI and underlying system core with its graphical output has not been completely implemented yet.

The studies showed, that the design support for testing the correctness of the structural concept was helpful. It had a different importance for all three engineers but was not reducing the usability of the system. The novice and unexperienced engineer reduced the rate of concepts that did not exhibit a global load transfer or violated constraints of the DIN 1045-1, whereas the experienced practitioner found it helpful that correctness of the structural concept was also checked in the background by the system. The design sup-

port of planning the design process prior designing, was experienced relatively to the engineer's level of control knowledge. For the experienced practitioner, the process control in form of the task structure dialog was not helpful, whereas for the less experienced engineer and the novice the control was important to improve their performance. The increased performance resulted from a focused design approach with less iterations. All test users felt supported in a flexible way, if the system allowed to modify the once designed concept in simple tasks.

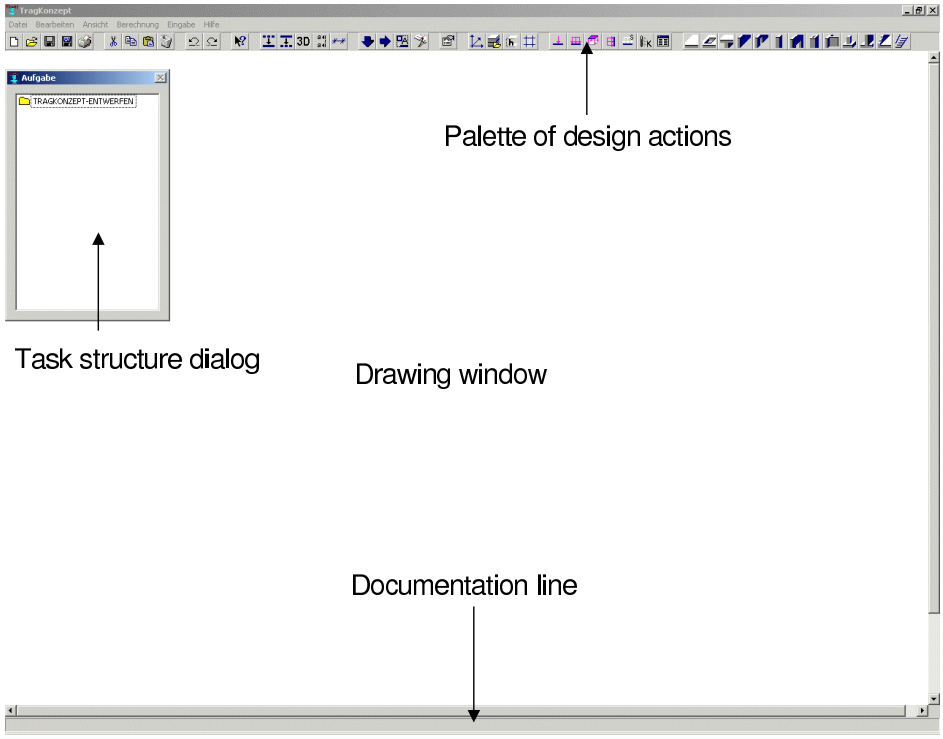


Figure 4.1: Main GUI components

The so far performed usability engineering steps resulted in a GUI design, depicted in figure 4.1. The main GUI components are the *drawing window*, the *palette of design actions*, the *task structure dialog*, and the *documentation line*. The buttons for the testing actions, e.g. vertical load transfer control, are also found at the palette. The *modal dialogs* for input about structural objects are not shown in the figure. Apart from the drawing window, a 3D-viewer shall be used to display the spatial concept under development. However, it is not implemented yet. Typical modal dialogs will be shown along a design session in Section 4.2.

The *drawing window* is an ordinary 2D window for graphical output. It displays the current structural concept and allows direct manipulation of structural elements and loads. Correct structural objects are displayed in blue, whereas incorrect ones that contradict the definitorial knowledge or need revision due to insufficient load transfer are displayed in red.

The *task structure dialog* is the control for the design process. It displays the task structure as decomposition tree with the subtasks dependencies on different task levels. A user can dynamically decompose the initial conceptual design task according to the current system state by clicking on an available task item in the tree. A task is highlighted in bold and called *focused* when the user clicks on the task item. It triggers the search process for applicable design methods M_{app} for compound tasks or design operators A_{app} for simple tasks, respectively. Since the subtasks have to be linearly-ordered the task decomposition tree is dynamically explored top-down. Furthermore, a task at the head of a tasklist is highlighted and set available in the decomposition tree, whereas tasks in the tasklist's tail are set unavailable. However, it is possible for the user to modify once entered design input for a focused simple task, which leads to local cycles in the tree. If a set of compound design methods is applicable, the engineer has to select a single one for further refinement. At the simple task level, design and testing action buttons from the palette become available for highlighted subtasks from which the user can select.

The *palette of design actions* is integrated in the application frame window. Action buttons in the palette become available for a focused simple task in the tree, if the corresponding design actions are applicable. The range of available design action buttons on the palette is dynamically adapted to the current system state by searching for the set of applicable design operators A_{app}. If a user selects a design action button, an input cycle for a structural object starts. It includes user input like points from the drawing window via mouse selection and from a displayed modal dialog. At the end of this cycle, a corresponding design action evolves the structural concept. Structural object input, which results in an inconsistent structural concept, is displayed in red and must be immediately revised. The button for load transfer control can be activated and deactivated. It uses the reasoning service of completeness testing to compute elements that do not exhibit a complete load transfer. In an active state, elements turn red in the drawing window that do not transfer correctly the loads to supporting members. In addition, a finish button allows the user to finish a focused task at any time. The finish action triggers a completeness test if required.

The *documentation line* outputs information about the design process and the structural concept. For example, after a design action has changed the structural concept, the documentation line displays information about the result of the consistency test. The system performs this test in the background. Or as another example, if the user has set up conflicts, he is prompted to reexamine these in the corresponding simple subtasks.

Modal dialogs for user input cover a set of object related design actions. The planning component computes, which of them are applicable to a focused element, and makes the corresponding widgets available on the dialog. For a structural element as example, the user can add an element, relate it to other structural objects, choose its type, and preliminary design it. However, at a certain system state, it may not be possible to size it, because first the type of element needs to be chosen or the internal forces have to be computed.

GUI elements like design action buttons on the palette and task items of the decomposition tree change dynamically their state depending on the current system state. To show how the system works, parts of it are explained therefore on the basis of the underlying formalism. The structural concept with its systems, elements, and loads is described by an evolving ABox and at selected points shown in a 3D-viewer. The task decomposition tree is directly given in the formalism instead of the task structure dialog. The tasklist t comprises the currently focused task and the remaining tasks at the same level but not tasks at higher levels. Tasks in front of a focused task are already removed but shown for clarity. If different task refinements or design actions are possible they are given as a set M_{app} or A_{app}. In these sets, test and finish actions are skipped for brevity. Furthermore, the add list is shown for synthesis actions, the delete list for delete actions, and both lists for modify actions, respectively. For readability, the concrete domain objects created in the binding lists are not written in the ABox and the operators.

4.2 Design Session with Prototype

For representation purposes, the same office building as in example 3 is employed. The origin of the coordinate system is located at the down left end of the building. The initial concept has first to be built up based on information in the drawings and the design brief. Information that is independent of a certain task instance like concrete classes, reinforcement grades, possible design criteria, foundation types etc. are stored in the initial ABox $\mathcal{A}_0$. They are related by *choose conditions* in the binding list of design actions to the structural concept, denoted CONCEPT. *Set conditions* denote user input.

The design process starts in the system state $\mathcal{S}_0 = (\mathcal{A}_0, t_0)$ when the user clicks on the highlighted initial task item

$$t_0 = [\textbf{designStructuralConcept}]$$

in the task decomposition tree of the task structure dialog. After the system has found the only applicable method

$$\begin{aligned} a_{P_1} \;=\; & \langle \mathsf{designStructuralConcept} \\ & \{\}, \\ & [\mathsf{enterBuilding}, \\ & \;\mathsf{checkFeasibility}, \\ & \;\mathsf{makeStructuralConcept}]\rangle, \end{aligned}$$

the top-level task t_0 expands to the list of ordered compound subtasks

$$t_1 = [\textbf{enterBuilding}, \mathsf{checkFeasibility}, \mathsf{makeStructuralConcept}]$$

in the decomposition tree. Since the subtasks are ordered, they have t be completed by the user in this order. In the current system state $\mathcal{S}_1 = (\mathcal{A}_0, t_1)$, the user starts to build-up the initial incomplete structural concept $\mathcal{I}_0$ represented by a consistent but incomplete ABox. In this subtask, the operators and methods have no additional preconditions to the consistency test because entering the building is a routine task. It consists of the assignment of the building parameters and design criteria, the existing structural elements with their geometry given by the architectural drawings, and applied loads on the structure. The load cases are self-weight, and variable actions like occupancy, wind, and snow. When the user focuses the subtask of entering the building, the system expands it by a planning action a_{P_2}, similar to a_{P_1}, to three ordered subtasks

$$t_2 = [\textbf{setParameter}, \mathsf{defineBuildingGeometry}, \mathsf{applyLoads}].$$

Now at the conceptual level, the engineer has to define actions on existing building elements. Three operators, denoted $A_{app} = \{a_{D_1}, a_{D_2}, a_{D_3}\}$, become applicable to the simple subtask for user choice in the action palette. The first operator

$$\begin{aligned} a_{D_1} \;=\; & \langle \mathsf{setParameter}, \\ & \{\mathsf{StateConsistentp}()\}, \\ & \{\mathsf{StructuralConcept}(?\mathsf{x}_1 \rightarrow \mathsf{CONCEPT}), \\ & \;\mathsf{chooseSoilClass}(?\mathsf{x}_2 \rightarrow a_0), \\ & \;\mathsf{chooseCriteria}(?\mathsf{x}_3 \rightarrow b_0), \end{aligned}$$

$$\begin{aligned}
&\mathsf{chooseFoundationType}(?\mathsf{x}_4 \rightarrow c_0),\\
&\mathsf{chooseConcreteClass}(?\mathsf{x}_5 \rightarrow d_0),\\
&\mathsf{chooseSteelGrade}(?\mathsf{x}_6 \rightarrow e_0),\\
&\mathsf{chooseExpositionClass}(?\mathsf{x}_7 \rightarrow f_0)\},\\
\{&\mathsf{hasSoilClass}(?\mathsf{x}_1, ?\mathsf{x}_2), \mathsf{hasCriteria}, (?\mathsf{x}_1, ?\mathsf{x}_3),\\
&\mathsf{hasFoundationType}, (?\mathsf{x}_1, ?\mathsf{x}_4),\\
&\mathsf{hasConcreteClass}(?\mathsf{x}_1, ?\mathsf{x}_5), \mathsf{hasSteelGrade}(?\mathsf{x}_1, ?\mathsf{x}_6),\\
&\mathsf{hasExpositionClass}(?\mathsf{x}_1, ?\mathsf{x}_7), \mathsf{Parameter}(?\mathsf{x}_1)\}\rangle
\end{aligned}$$

can be used to *assign building parameters*. Objects that are already in the initial ABox are denoted by a subscript. The second applicable operator is

$$\begin{aligned}
a_{D_2} \quad = \quad &\langle \mathsf{setParameter},\\
&\{\mathsf{StateConsistentp}()\},\\
&\{\mathsf{StructuralConcept}(?\mathsf{x}_1 \rightarrow \mathsf{CONCEPT}),\\
&\ \mathsf{setFunctionalGrid}_{x_m}(?\mathsf{x}_2 \rightarrow a), \mathsf{setFunctionalGrid}_{y_m}(?\mathsf{x}_3 \rightarrow b)\},\\
&\{\mathsf{hasFunctionalGrid}_{x_m}(?\mathsf{x}_1, ?\mathsf{x}_2), \mathsf{hasFunctionalGrid}_{y_m}(?\mathsf{x}_1, ?\mathsf{x}_3),\\
&\ \mathsf{Grid}(?\mathsf{x}_1)\}\rangle
\end{aligned}$$

to *choose a functional grid*. The third one, denoted a_{D_3}, is to *define storey heights* and *occupancies*. When the user for example selects the operator a_{D_1} a modal dialog, shown in 4.2, is opened.

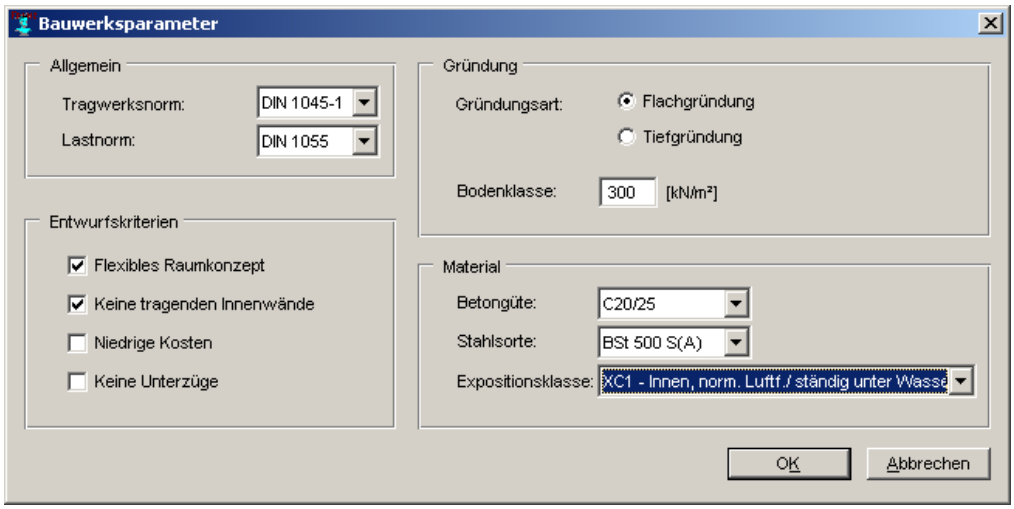


Figure 4.2: Building parameter dialog

After the user has entered the building parameter values in the figure, the system substitutes the variables in the operator's add list and transforms the current state $\mathcal{A}_0$ into the successor ABox

$$\begin{aligned}\mathcal{A}_1 \;=\; \{&\mathsf{hasSoilClass}(\mathsf{CONCEPT}, \mathsf{GE}),\\ &\mathsf{hasCriteria}, (\mathsf{CONCEPT}, \mathsf{SPACE/NOWALLS}),\\ &\mathsf{hasFoundationType}, (\mathsf{CONCEPT}, \mathsf{SHALLOW}),\\ &\mathsf{hasConcreteClass}(\mathsf{CONCEPT}, \mathsf{C20/25}),\\ &\mathsf{hasSteelGrade}(\mathsf{CONCEPT}, \mathsf{B500S}),\\ &\mathsf{hasExpositionClass}(\mathsf{CONCEPT}, \mathsf{XC3}),\\ &\mathsf{Parameter}(\mathsf{CONCEPT}), \ldots\},\end{aligned}$$

where GE stands for a good soil class, SPACE/NOWALLS for a flexible room concept without internal walls, and SHALLOW for a foundation with single and line footings, respectively. These abstract constraints are later used to select suitable element types. The user adds the functional grid to the concept by the design action a_{D_2} with the ground effect $\mathsf{hasGrid}(\mathsf{CONCEPT}, \mathsf{GRID})$ and $\mathsf{FunctionalUnit}_m(\mathsf{GRID}, \mathsf{a}_1)$ with $\mathsf{a}_1 = 5.0$ (because the two functional unit lengths are the same they are abbreviated by one). This action results in

$$\mathcal{A}_2 \;= \mathcal{A}_1 \cup \{\mathsf{hasGrid}(\mathsf{CONCEPT}, \mathsf{GRID}), \mathsf{FunctionalUnit}_m(\mathsf{GRID}, \mathsf{a}_1)\}.$$

Finally, he chooses the operator a_{D_3} to define storey occupancies and storey heights, represented in $\mathcal{A}_3$ by $\mathsf{hasPart}(\mathsf{CONCEPT}, \mathsf{G1})$, $\mathsf{StoreyHeight}_m(\mathsf{G1}, \mathsf{h}_1)$, $\mathsf{hasOccupancy}(\mathsf{G1}, \mathsf{p}_1)$, and $\mathsf{Shop}(\mathsf{p}_1)$ for the ground storey for example.

The user can also skip design actions, since a building does not always have a functional grid or changing storey heights. When the user wants to change or delete his entered input, he can do so by the "modify" and "delete" operators. Corresponding modify operators $a_{D_4}, a_{D_5}, a_{D_6}$ exist for each of the three operators. A delete operator a_{D_7} exists only for the grid action because the other design information has to exist and cannot hence be deleted. A modification of the building parameters is for example possible, if an assertion of the current design state matches the condition $\mathsf{Parameter}(?\mathsf{x}_1)$ in the precondition of the modify operator with name $name(a_{D_4}) = \mathsf{setParameter}$. It works for the operators a_{D_5} and a_{D_6} correspondingly. A delete action like a_{D_5} removes ground conditions from the current ABox.

After every design step, the design state is tested for consistency. When the user for example finishes the parameter subtask, $\mathcal{A}_3$ is checked on an axiom of the definitorial conceptual knowledge $\mathcal{T}$ for consistency.

$$
\begin{aligned}
\mathsf{BuildingParameter} \;\doteq\; & \exists_{=1}\mathsf{hasCriteria}.\top \sqcap \\
& \exists_{=1}\mathsf{hasFoundationType}.\top \sqcap \\
& \exists \mathsf{ConcreteClass.Strength}_{N/mm^2}.\leq_{100} \sqcap \\
& \exists \mathsf{hasPart}.(\exists \mathsf{hasPart.Slab} \sqcap \\
& \exists \mathsf{supports}.(\mathsf{ConcentratedLoad} \sqcap \\
& \qquad \neg\exists \mathsf{hasValue}_{kN}.>_7))) \sqcap \\
& \forall \mathsf{hasPart}.(\mathsf{hasOccupancy}.(\mathsf{Residence} \sqcup \mathsf{Office} \sqcup \\
& \qquad \mathsf{Shop} \sqcup \mathsf{Accumulation}))
\end{aligned}
$$

If for example the user defines distributed loads that are larger than 5 kN/m^2 or a concentrated load that is larger than 7 kN, which lies outside the scope of ordinary building structures according to the DIN 1045-1, the system notices that and outputs this information on the documentation line. In the example, the building parameters are consistent and the user can continue with its design. Otherwise, he has to modify his input until it becomes consistent. The finish action adds the assertion $\mathsf{setParameterOK}(\mathsf{CONCEPT})$ by the design action a_{D_8} to set the focus on the subsequent subtask.

Up to now, the actions $a_{P_1}, a_{P_2}, a_{D_1}, a_{D_2}, a_{D_3}$, and a_{D_8} have been appended to the empty solution plan resulting in $\mathcal{P}_1 = [a_{D_8}, a_{D_3}, a_{D_2}, a_{D_1}, a_{P_2}, a_{P_1}]$ and the system state $\mathcal{S}_6 = (\mathcal{A}_4, t_2)$. If the user has modified the grid, for example, the solution plan includes an additional action a_{D_5} somewhere between a_{D_8} and a_{D_2}. When the user clicks on the highlighted subtask item in

$$t_2 = [\mathsf{setParameter}, \mathbf{defineBuildingGeometry}, \mathsf{applyLoads}].$$

in the decomposition tree, the system searches for applicable methods and finds the method

$$
\begin{aligned}
a_{P_3} \;=\; & \langle \mathsf{defineBuildingGeometry} \\
& \{\mathsf{StateConsistentp}(), \mathsf{setParameterOK}(\mathsf{CONCEPT})\}, \\
& [\mathsf{makeStoreyGeometry}, \ldots, \\
& \;\mathsf{makeLateralSystem}]\rangle,
\end{aligned}
$$

which expands the focused subtask into the tasklist

$$t_3 = [\mathbf{makeStoreyGeometry}, \ldots, \mathsf{makeLateralSystem}]$$

to define the element geometries at each storey and to set up stabilizing elements. As an example, the dialog shown in figure 4.3 is referenced by the

first subtask of t_3 for which two different design operators, one for *slabs* and one for *walls*, have become applicable, denoted by $A_{app} = \{a_{D_9}, a_{D_{10}}\}$. Walls that are located only at a single storey do not exist in this example.

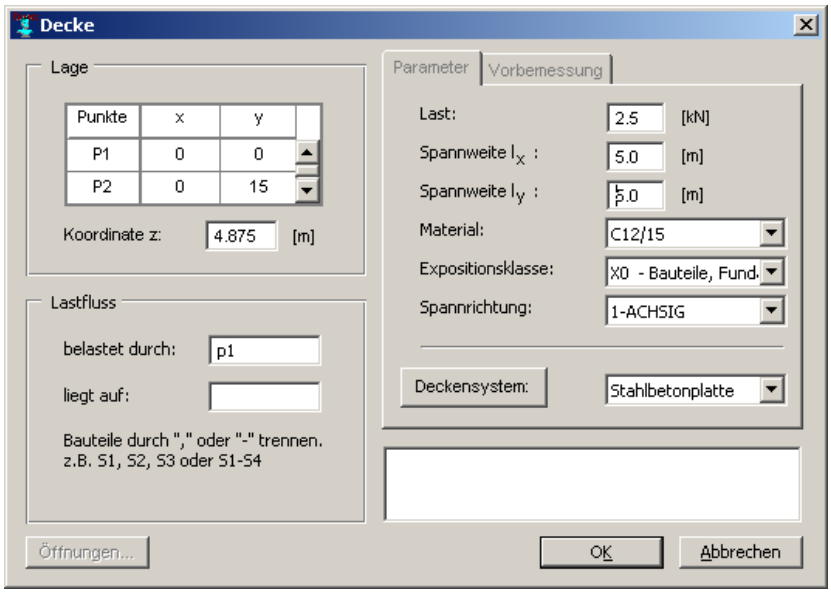


Figure 4.3: Slab dialog

After the user has entered the values for the geometry by selecting points in the drawing window, the design action

$$
\begin{aligned}
a_{D_9} \quad = \quad & \langle \mathsf{makeBuildingGeometry}, \\
& \{\mathsf{StateConsistentp}()\}, \\
& \{\mathsf{Storey}(\mathsf{x}_1 \rightarrow \mathsf{G1}), \mathsf{Slab}(\mathsf{x}_2 \rightarrow \mathsf{SL1}), \\
& \ \mathsf{setGeometry}(\mathsf{GE}_1 \rightarrow [(0.0, 0.0), (15.0, 0.0)(15.0, -15.0), \\
& \qquad\qquad (30.0, -15.0), (30.0, 9.0)(28.0, 9.0), \\
& \qquad\qquad (24.0, 13.0), (24.0, 15.0), (0.0, 15.0)]), \\
& \ \mathsf{setKo}_{z_m}(\mathsf{z}_1 \rightarrow 4.875), \\
& \ \mathsf{setLoad}(\mathsf{p}_1 \rightarrow \mathsf{6B}), \\
& \ \mathsf{chooseSupportElements}(\mathsf{lst}_1 \rightarrow \mathsf{nil}), \\
& \ \mathsf{chooseLoadElements}(\mathsf{lst}_2 \rightarrow [\mathsf{p}_1])\}, \\
& \{\mathsf{Slab}(\mathsf{SL1}), \\
& \ \mathsf{hasPart}(\mathsf{G1}, \mathsf{SL1}), \mathsf{hasCoordinate}(\mathsf{SL1}, \mathsf{z}_1) \\
& \ \mathsf{hasGeomerty}(\mathsf{SL1}, \mathsf{GE}_1), \mathsf{loads}(\mathsf{p}_1, \mathsf{SL1})\}\rangle
\end{aligned}
$$

transforms the current ABox into $\mathcal{A}_5$ where bound variables are written without question mark. In this dialog, the user can only enter loading or supporting elements, which have been previously defined. In the example, the occupancy load p_1 is displayed because it loads the slab. The items for *selecting a slab type* and *sizing it by a rule of thumb* are not available at the tab-control, because their corresponding design operators $name(a_{D_{15}}) = \mathsf{selectSlabType}$ and $name(a_{D_{16}}) = \mathsf{SizeSlab}$ are not applicable to the focused subtask. If the user tries despite to select one of them, the system outputs at the documentation line, that selecting a slab type is part of the subtask checkFeasibility and that sizing the slab requires a chosen slab type plus additional information. After the user has entered in five steps the other slabs for each storey by copy actions, the intermediate plan is now given by $\mathcal{P}_2 = [a_{D_{19}}, a_{D_9}, \ldots, a_{P_3}, \mathcal{P}_1]$ with $a_{D_{19}}$ a finish action. In the system state $\mathcal{S}_{19} = (\mathcal{A}_{16}, t_3)$,

$$t_3 = [\mathsf{makeStoreyGeometry}, \ldots, \mathbf{makeLateralSystem}]$$

makes the design operators $A_{app} = \{a_{D_{20}}, a_{D_{21}}\}$ for *shear walls* and *cores* applicable. The user adds the two sanitary cores located at either ends of the building. He could also select existing shear walls as stabilizing elements. As an example, consider the core dialog shown in figure 4.4.

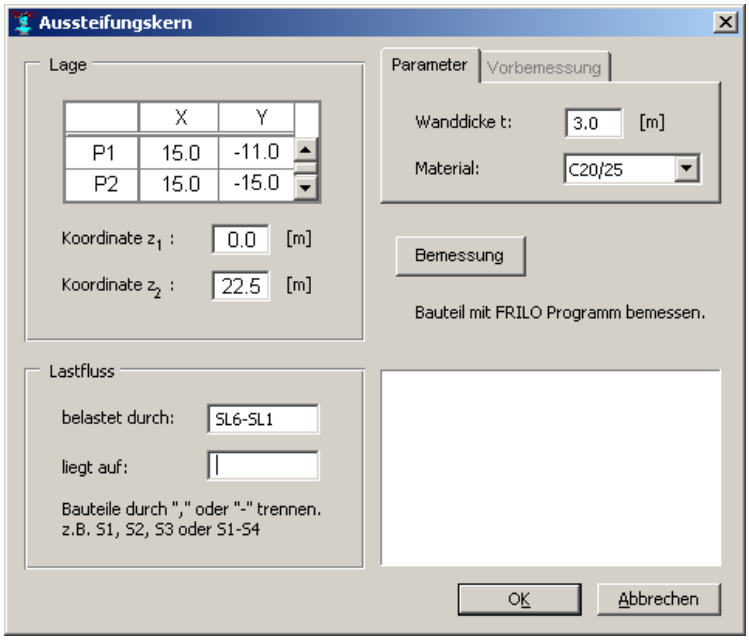


Figure 4.4: Core dialog

The corresponding design action is used to add the right sanitary core to the lateral stability system.

$$
\begin{aligned}
a_{D_{21}} \;=\; & \langle \mathsf{makeStabilitySystem}, \\
& \{\mathsf{StateConsistentp}(), \mathsf{StoreyGeometryOK}(6)\}, \\
& \{\mathsf{StabilitySystem}(\mathsf{x}_1 \rightarrow \mathsf{LS}), \mathsf{Core}(\mathsf{x}_2 \rightarrow \mathsf{CO1}), \\
& \ \mathsf{setGeometry}(\mathsf{GE}_2 \rightarrow [(15.0, -11.0), (15.0, -15.0), \\
& \qquad\qquad (19.5, -15.0), (19.5, -11.0)]), \\
& \ \mathsf{setKo}_{z_m}(\mathsf{z}_1 \rightarrow 0.0), \mathsf{setKo}_{z_m}(\mathsf{z}_2 \rightarrow 22.5), \\
& \ \mathsf{setCoreThickness}(\mathsf{t}_1 \rightarrow 0.3), \\
& \ \mathsf{chooseConcrete}(\mathsf{RC} \rightarrow \mathsf{C30}_0), \\
& \ \mathsf{chooseSupportElements}(\mathsf{lst}_1 \rightarrow \mathsf{nil}), \\
& \ \mathsf{chooseLoadElements}(\mathsf{lst}_2 \rightarrow [\mathsf{SL1}, \mathsf{SL2}, \mathsf{SL3}, \mathsf{SL4}, \mathsf{SL5}, \mathsf{SL6}]), \\
& \ \mathsf{calculateSelfWeight}_{kN/m}(\mathsf{g}_{C_1} = 25 \cdot 0.3 \cdot 2 \cdot (4.0 + 3.0))\}, \\
& \{\mathsf{Core}(\mathsf{CO1}), \mathsf{hasPart}(\mathsf{LS}, \mathsf{CO1}), \\
& \ \mathsf{hasGeomerty}(\mathsf{CO1}, \mathsf{GE}_2), \mathsf{hasMaterial}(\mathsf{CO1}, \mathsf{C30}), \\
& \ \mathsf{hasCoordinate}(\mathsf{CO1}, \mathsf{z}_1), \mathsf{hasCoordinate}(\mathsf{CO1}, \mathsf{z}_2), \\
& \ \mathsf{Thickness}_m(\mathsf{CO1}, \mathsf{t}_1), \\
& \ \mathsf{supports}(\mathsf{CO1}, \mathsf{SL1}), \ldots, \\
& \ \mathsf{SelfWeight}_{kN/m}(\mathsf{CO1}, \mathsf{g}_{C_1})\}\rangle
\end{aligned}
$$

The user indicates by the design action $name(a_{D_{28}}) = \mathsf{makeStabilitySystem}$ that he has finished the initial stability system LS given by the architectural design. The system computes first in the binding list of $a_{D_{28}}$, whether or not the cores are symmetrically located. In the example, $\neg\mathsf{Symmetric}(\mathsf{LS})$ is deduced. The system outputs on the documentation line that the user should provide additional shear walls in the subtask checkFeasibility and adds the assertion$\neg\mathsf{Symmetric}(\mathsf{LS})$ to the current ABox. Now, the algorithm checks in the consistency test if the object $\mathsf{StabilitySystem}(\mathsf{LS})$ is at least consistent on the inclusion axiom

$$
\begin{aligned}
\mathsf{StabilitySystem} \;\sqsubseteq\; & \exists_{=3}\, \mathsf{hasPart}.\, \mathsf{ShearWall} \;\sqcup \\
& \exists_{\geq 2}\, \mathsf{hasPart}.\, \mathsf{Core} \;\sqcup \\
& (\exists_{=1}\, \mathsf{hasPart}.\, \mathsf{Core} \;\sqcap \\
& \exists_{=2}\, \mathsf{hasPart}.\, \mathsf{ShearWall}).
\end{aligned}
$$

The algorithm only tests if design assertions about the stability system given by the ABox part

$$\begin{aligned}\mathcal{A}_{19} \;=\; \{&\ldots,\\ &\mathsf{StabilitySystem}(\mathsf{LS}),\\ &\mathsf{hasPart}(\mathsf{LS},\mathsf{CO1}),\\ &\mathsf{hasPart}(\mathsf{LS},\mathsf{CO2}),\\ &\mathsf{Core}(\mathsf{CO1}),\\ &\mathsf{Core}(\mathsf{CO2}),\ldots\},\end{aligned}$$

contradict the inclusion, which is not the case in this example. This test however is not a sufficient condition to provide lateral stability, indicated by the inclusion axiom. Since the shafts are not symmetrically placed, which results in a twist around the shear center, additional shear walls have to be provided. The symmetry assertion from above ensures that the user will later add further stabilizing elements in the design process.

The subtask makeStabilitySystem results in $\mathcal{P}_3 = [a_{D_{28}}, a_{D_{21}}, a_{D_{21}}, \mathcal{P}_2]$ and a design state $\mathcal{S}_{22} = (\mathcal{A}_{19}, t_2)$ with the task applyLoads focused. It includes design information about the specific building parameter, occupancies, the grid, the six storeys with their slab geometry and the two stiffening sanitary cores as well as the support and compositional relations among the structural objects.

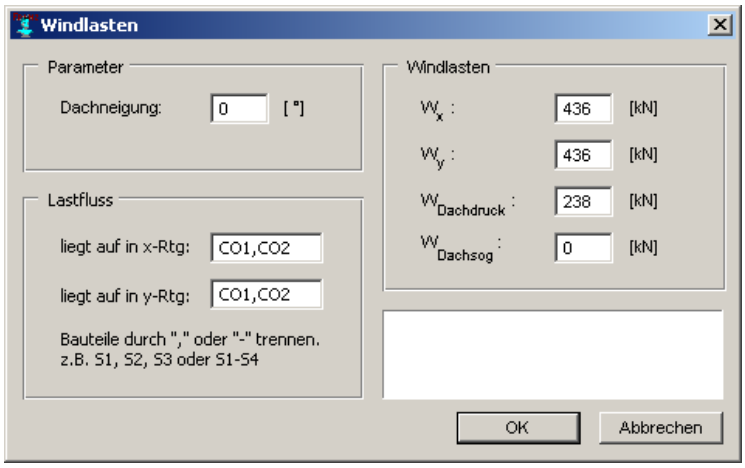


Figure 4.5: Wind load dialog

After the system has found an applicable method a_{P_4} for the next subtask in

$$t_2 = [\mathsf{makeParameter}, \mathsf{defineBuildingGeometry}, \mathbf{applyLoads}]$$

the focused subtask expands to the tasklist [applyLoad] from which operators $A_{app} = \{a_{D_{29}}, a_{D_{30}}, a_{D_{31}}, a_{D_{32}}, a_{D_{33}}\}$ are referenced to define other loads on the structure like *concentrated loads*, *linear loads*, *distributed loads*, *snow loads*, and *wind loads*. In the example, only wind loads and snow loads are applied to the building. As an example, consider the design operator $a_{D_{33}}$ for wind loading, which corresponds to the dialog shown in figure 4.5. After the user has entered the values and closed the dialog, the system calculates the resultant wind forces WL1, WL1 and applies the design action

$$
\begin{aligned}
a_{D_{33}} \;=\; & \langle \mathsf{applyLoad}, \\
& \{\mathsf{StateConsistentp}(), \\
& \{\mathsf{WindLoad}(\mathsf{x}_1 \rightarrow \mathsf{WL1}), \mathsf{Windload}(\mathsf{x}_2 \rightarrow \mathsf{WL2}), \\
& \ \mathsf{setRoofAngle}(\alpha \rightarrow 0.0^\circ), \\
& \ \mathsf{chooseSupportElements}(\mathsf{lst}_1 \rightarrow [\mathsf{LS}]), \\
& \ \mathsf{calculateForce}_{kN}(\mathsf{FW}_x = (0.8 + 0.5) \cdot 30 \cdot 22.5), \\
& \ \mathsf{calculateForce}_{kN}(\mathsf{FW}_y = (0.8 + 0.5) \cdot 30 \cdot 22.5)\}, \\
& \{\mathsf{WindLoad}(\mathsf{WL1}), \mathsf{WindLoad}(\mathsf{WL2}), \\
& \ \mathsf{loads}(\mathsf{WL1}, \mathsf{CO1}), \mathsf{loads}(\mathsf{WL1}, \mathsf{CO2}), \\
& \ \mathsf{loads}(\mathsf{WL2}, \mathsf{CO1}), \mathsf{loads}(\mathsf{WL1}, \mathsf{CO2}), \\
& \ \mathsf{hasValue}_{kN}(\mathsf{WL1}, \mathsf{FW}_x), \\
& \ \mathsf{hasValue}_{kN}(\mathsf{WL2}, \mathsf{FW}_y)\}\rangle.
\end{aligned}
$$

Thereafter, he defines a snow load by an action $a_{D_{32}}$ and finishes the focused subtask by the operator $name(a_{D_{44}}) = \mathsf{applyLoad}$ that adds the assertion $\mathsf{appliedLoads}(\mathsf{CONCEPT})$ to the ABox. Other loads are not applied to the structure in this example.

The ABox $\mathcal{A}_{22}$ resulting from the subsequent actions $a_{D_{33}}$ and $a_{D_{32}}$ is checked for consistency, again. Since it the algorithm returns consistent on the conceptual knowledge, the initial structural concept $\mathcal{I}_0$ is reached via the design plan $\mathcal{P}_4 = [a_{D_{44}}, a_{D_{32}}, a_{D_{33}}, a_{P_4}, \mathcal{P}_3]$ in the system state $\mathcal{S}_{26} = (\mathcal{A}_{22}, t_1)$ with checkFeasibility focused. The initial structural concept as interpretation can be constructed from the ABox $\mathcal{A}_{22}$ and includes the structural objects, the load transfer and compositional relations among structural objects, and the constraints. After all objects have been realized into their most specific concepts in the TBox $\mathcal{T}$ the interpretation $\mathcal{I}_0 = \langle \Delta_{\mathcal{I}_0}, \cdot^{\mathcal{I}} \rangle$ is given as follows for which $\cdot^{\mathcal{I}}$ corresponds to $R_0^{\mathcal{I}}$ and $F_0^{\mathcal{I}}$:

$$
\begin{aligned}
\Delta_{\mathcal{I}_0} = \{&\mathsf{CONCEPT}, \mathsf{G1}, \ldots, \mathsf{G6}, \mathsf{SL1}, \ldots, \mathsf{SL6},\\
&\mathsf{LS}, \mathsf{CO1}, \mathsf{CO2},\\
&\mathsf{SPACE/NOWALLS}, \mathsf{GE}, \mathsf{SHALLOW}, \mathsf{WL1}, \mathsf{WL2},\\
&\mathsf{g}_1, \ldots, \mathsf{g}_7, \mathsf{p}_1, \ldots, \mathsf{p}_7, \mathsf{g}_{C_1}, \mathsf{g}_{C_2}, \mathsf{s}\},\\
\mathsf{StructuralObject}^{\mathcal{I}_0} = {}&\mathsf{Load}^{\mathcal{I}} \cup \mathsf{Element}^{\mathcal{I}_0} \text{ with all } x \in \mathsf{Load}^{\mathcal{I}_0} \text{ maximal},\\
\mathsf{Load}^{\mathcal{I}_0} = {}&\{\mathsf{WL1}, \mathsf{WL2}, \mathsf{p}_1, \ldots, \mathsf{p}_7, \mathsf{g}_1, \ldots, \mathsf{g}_7, \mathsf{g}_{C_1}, \mathsf{g}_{C_2}, \mathsf{s}\},\\
\mathsf{Element}^{\mathcal{I}_0} = {}&\{\mathsf{SL1}, \ldots, \mathsf{SL6}, \mathsf{CO1}, \mathsf{CO2}\},\\
\mathsf{StoreySystem}^{\mathcal{I}_0} = {}&\{\mathsf{G1}, \ldots, \mathsf{G6}\},\\
\mathsf{System}^{\mathcal{I}_0} = {}&\{\mathsf{G1}, \ldots, \mathsf{G6}, \mathsf{LS}\},\\
\mathsf{Designable}^{\mathcal{I}_0} = {}&\emptyset \text{ and } \mathsf{Stable}^{\mathcal{I}_0} = \emptyset,\\
\mathsf{Constraint}^{\mathcal{I}_0} = {}&\{\mathsf{SPACE/NOWALLS}, \mathsf{GE}, \mathsf{SHALLOW}\},\\
\mathsf{supports}^{\mathcal{I}_0} = {}&\{(\mathsf{LS}, \mathsf{WL1}), (\mathsf{LS}, \mathsf{WL2}), (\mathsf{CO1}, \mathsf{p}_2), \ldots, (\mathsf{CO1}, \mathsf{p}_6),\\
&(\mathsf{CO2}, \mathsf{p}_2), \ldots, (\mathsf{CO2}, \mathsf{p}_7),\\
&(\mathsf{SL1}, \mathsf{p}_2), \ldots, (\mathsf{SL6}, \mathsf{p}_7),\\
&(\mathsf{CO1}, \mathsf{g}_{C_1}), \ldots, (\mathsf{CO2}, \mathsf{g}_{C_2}),\\
&(\mathsf{SL1}, \mathsf{g}_2), \ldots, (\mathsf{SL6}, \mathsf{g}_7), (\mathsf{SL6}, \mathsf{s})\},\\
\mathsf{hasPart}^{\mathcal{I}_0} = {}&\{(\mathsf{CONCEPT}, \mathsf{G1}), \ldots, (\mathsf{CONCEPT}, \mathsf{G6}),\\
&(\mathsf{CONCEPT}, \mathsf{LS}), (\mathsf{LS}, \mathsf{CO1}), (\mathsf{LS}, \mathsf{CO2}),\\
&(\mathsf{G1}, \mathsf{SL1}), \ldots, (\mathsf{G6}, \mathsf{SL6}), \}, \text{ and}\\
\mathsf{hasConstraint}^{\mathcal{I}_0} = {}&\{(\mathsf{CONCEPT}, \mathsf{SPACE/NOWALLS}), (\mathsf{CONCEPT}, \mathsf{GE}),\\
&(\mathsf{CONCEPT}, \mathsf{SHALLOW})\}.
\end{aligned}
$$

The domain comprises the incomplete concept CONCEPT, the storey systems $\mathsf{G1}-\mathsf{G6}$, the slab systems $\mathsf{SL1}-\mathsf{SL6}$, the lateral stability system LS, the cores CO1,CO2, the loads g_{1-7},p_{1-7}, the snow load s, and the wind loads WL1,WL2. The loads applied to the structure are maximal in the set of structural objects according to the support relations. The set of support relations does not order the structural objects totally since no minimal objects exist. The elements are not designed yet and the building is not stable because of missing lateral elements. The elements are added to the subset of designable elements if they have been approximately sized by rules of thumb. Therefore, the partial functions $\mathit{state\text{-}variable}^{\mathcal{I}_0}$ and $\mathit{parameter}^{\mathcal{I}_0}$ are not used, yet. The initial structural concept is shown in a 3D-view as in figure 4.6.

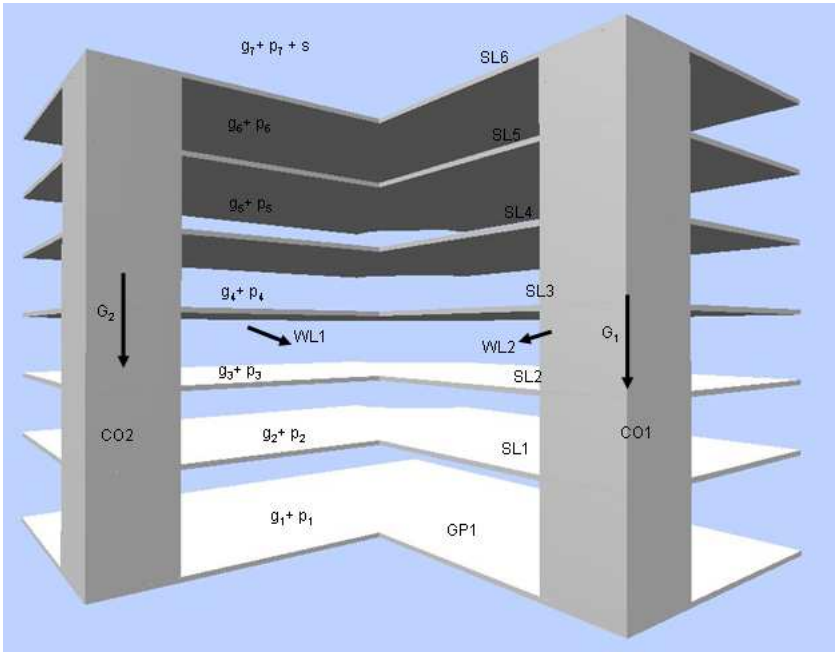


Figure 4.6: Initial structural concept

In the second major compound subtask of

$$t_1 = [\mathsf{enterBuilding}, \mathbf{checkFeasibility}, \mathsf{makeStructuralConcept}]$$

the engineer wants to establish a stability system, find the conflicting points because of load transfer discontinuities, and choose an appropriate slab type. Therefore, the system finds a method

$$\begin{aligned} a_{P_5} \quad = \quad & \langle \mathsf{checkFeasibility} \\ & \{\mathsf{StateConsistentp}(), \mathsf{appliedLoads}(?\mathsf{x}_1), \neg\mathsf{symmetric}(?\mathsf{x}_2)\} \\ & [\mathsf{makeStabilitySystem}, \\ & \ \mathsf{checkConflicts}, \\ & \ \mathsf{chooseSlabSystemType}]\rangle, \end{aligned}$$

which is applicable under the substitution $\tau = \{?\mathsf{x}_1/\mathsf{CONCEPT}, ?\mathsf{x}_2/\mathsf{LS}\}$ to $\mathcal{S}_{26}$ because $name(a_{P_5}) = \mathsf{checkFeasibility}$ and

$$\{\mathsf{StateConsistentp}(), \mathsf{appliedLoads}(\mathsf{CONCEPT}), \neg\mathsf{symmetric}(\mathsf{LS})\} \subseteq \mathcal{A}_{22}.$$

This task requests from the user to define additional shear walls due to the unsymmetric arrangement of the cores. This compound subtask expands thus

by the method a_{P_5} to

$$t_4 = [\textbf{makeStabilitySystem}, \mathsf{checkConflicts}, \mathsf{selectSlabType}].$$

For the first subtask, the make operators with their corresponding modify and delete operators $A_{app} = \{a_{D_{20}}, \ldots a_{D_{25}}\}$ become applicable again. The system would find another method in which this subtask is missing, if the assertion $\neg\mathsf{symmetric}(\mathsf{LS})$ was not in the design state. After the user has entered two more shear walls $\mathsf{SW1}, \mathsf{SW2}$ and finished the subtask by $a_{D_{28}}$, the design information about the shear walls and the assertion $\mathsf{hasPart}(\mathsf{CONCEPT}, \mathsf{LS})$ about the system are inserted to the current ABox. The system has again computed in the binding list of the finish action, whether or not the cores and shear walls are symmetrically located, and has deleted the assertion $\neg\mathsf{Symmetric}(\mathsf{LS})$ and added the assertion $\mathsf{Symmetric}(\mathsf{LS})$ to the ABox in the example.

In the subsequent subtask **checkConflicts**, the engineer has to identify any load path discontinuities due to changing storey elevations or local load transfer disturbances. This task is, however, not implemented yet and cannot be supported by the system. In the example, the engineer has to find the conflict, which arises from the shear walls at the stairwell that must not be located at the ground storey. Therefore, a local solution will later be required. When the engineer indicates this conflict by an action of type $name = \mathsf{setConflict}(a_{D_{46}})$, the assertion $\mathsf{conflict}(\mathsf{SW1})$ as ground effect is added to the current ABox. After he has finished the subtask by the action $a_{D_{48}}$ he continues with the design process.

The next subtask **selectSlabType** of t_4 is to find a suitable slab system which can be performed by a relational data base query. In the data base, all possible slab types are stored in terms of span length ratio, load magnitude, maximum span length, and suitable building criteria. The design action

$$\begin{aligned} a_{D_{18}} \;=\; & \langle \mathsf{selectSlabType}, \\ & \{\mathsf{StateConsistentp}(), \mathsf{hasGrid}(\mathsf{CONCEPT}, \mathsf{GRID}), \\ & \;\mathsf{hasPart}(\mathsf{G6}, \mathsf{SL6})\}, \\ & \{\mathsf{selectSlabType}_{(5.0,5.0,3.75,\mathsf{SPACE/NOWALLS})}(\mathsf{SL6} \rightarrow \mathsf{FlatSlab}), \\ & \;\mathsf{SetSpanLength}(\mathsf{l} \rightarrow 5.0)\}, \\ & \{\mathsf{FlatSlab}(\mathsf{SL6}), \mathsf{SpanLength}_{max}(\mathsf{SL6}, \mathsf{l})\} \end{aligned}$$

realizes this data base request, where the required parameters are written in subscript. The user can select one of the slab type entries found for his slab design. In the example, the user decides to use a flat slab system with a

span length of 5 m, asserted by $\mathsf{FlatSlab}(\mathsf{SL6})$ and $\mathsf{SpanLength}_{max}(\mathsf{SL6}, \mathsf{l})$. If a feasible slab system cannot be found, the user has to change the input values for the data base request. The user finishes the subtask checkFeasibility by the design action $a_{D_{49}}$ which adds $\mathsf{checkFeasibilityOK}(\mathsf{CONCEPT})$ to the current ABox. The design process results in the system state $\mathcal{S}_{34} = (\mathcal{A}_{29}, t_1)$ with MakeStructuralConcept focused and the sequence $\mathcal{P}_4 = [a_{D_{49}}, a_{D_{18}}, \ldots, a_{P_5}, \mathcal{P}_3]$. The consistent ABox

$$\begin{aligned}\mathcal{A}_{30} \;=\; \{\ldots,\;& \mathsf{Symmetric}(\mathsf{LS}),\\ & \mathsf{hasPart}(\mathsf{LS}, \mathsf{SW1}),\\ & \mathsf{hasPart}(\mathsf{LS}, \mathsf{SW2}),\\ & \mathsf{ShearWall}(\mathsf{SW1}),\\ & \mathsf{ShearWall}(\mathsf{SW2}),\\ & \mathsf{conflict}(\mathsf{SW1}),\\ & \mathsf{FlatSlab}(\mathsf{SL1}), \ldots\\ & \mathsf{checkFeasibilityOK}(\mathsf{CONCEPT})\},\end{aligned}$$

for which only the newly inserted structural objects and relations are given. If the user controls the vertical load transfer at this point by a testing action a_{T_1}, the system identifies that the shear walls SW1 and SW2 are not supported and the concept is incomplete. A load on the shear wall is defined by the concept

$$\mathsf{Load} \equiv \exists \mathsf{SelfWeight}_{kN}.\top \sqcup \mathsf{TurningMoment}$$

or another structural element like a slab. Suitable adjacent elements to support the wall are a line footing or two columns. A pair of completeness axioms from the CBox $\mathcal{C}$

$$\mathsf{LoadedShearWall} \equiv \mathsf{ShearWall} \sqcap \exists \mathsf{supports}.(\mathsf{Load} \sqcup \mathsf{Slab})$$

$$\begin{aligned}\mathsf{DesignedShearWall} \equiv\; & \mathsf{ShearWall} \sqcap \mathsf{Designable} \sqcap\\ & (\exists_{n=2}\mathsf{loads}.\mathsf{Column} \sqcup \exists_{n=1}\mathsf{loads}.\mathsf{LineFooting})\end{aligned}$$

are used to compute the incomplete shear walls.

On the basis of the performed feasibility check, the system plans how to proceed in the last compound subtask of

$$t_1 = [\mathsf{enterBuilding}, \mathsf{checkFeasibility}, \textbf{makeStructuralConcept}].$$

The prototype suggests two possible task decompositions $M_{app} = \{a_{P_6}, a_{P_7}\}$ according to the current state. In the example, it finds the applicable instantiated method

$$\begin{aligned} a_{P_6} \;=\; & \langle \mathsf{makeStructuralConcept}, \\ & \{\mathsf{StateConsistentp}(), \mathsf{hasStabilitySystem}(\mathsf{CONCEPT}, \mathsf{LS}), \\ & \;\mathsf{checkFeasibilityOK}(\mathsf{CONCEPT})\}, \\ & [\mathsf{makeVerticalSystem}, \\ & \;\mathsf{sizeStructuralElements}, \\ & \;\mathsf{makeFoundation}, \\ & \;\mathsf{calculateLability}]\rangle \end{aligned}$$

and the method a_{P_7} for which makeFoundation and calculateLability have the opposite order than in a_{P_6}. The engineer selects the former one. The subtask **makeVerticalSystem** from the tasklist of a_{P_6} expands to

$$t_5 = [\textbf{makeVerticalStoreySystem}, \ldots]$$

by which the engineer can start to design the six vertical storey systems from top to bottom. At the storey level, the focused subtask is decomposed by

$$\begin{aligned} a_{P_8} \;=\; & \langle \mathsf{makeVerticalStoreySystem}, \\ & \{\mathsf{FlatSlab}(\mathsf{SL6})\}, \\ & [\mathsf{makeColumns}, \mathsf{sizeFlatSlab}]\rangle \end{aligned}$$

for which the tasklist $t_6 = [\textbf{makeColumns}, \mathsf{sizeFlatSlab}]$ has to be elaborated in this order because the slab design is non-critical for the chosen span length.

The subtask **makeColumns** expands by a method from the knowledge base to a tasklist which references an operator to define *columns*. The user has thus first to define the columns before he can size the flat slab. This is done by selecting points via mouse click in the drawing window for the column location and user input via the column dialog, shown in 4.7.

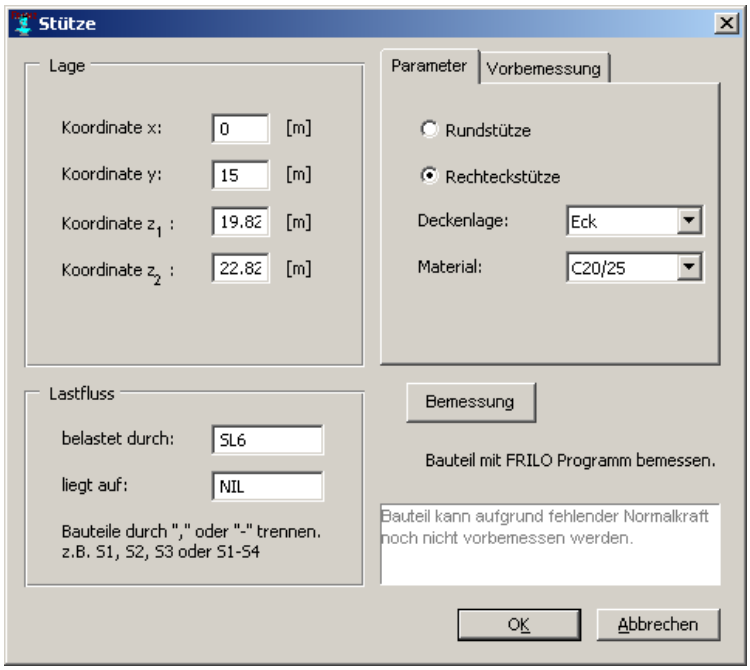


Figure 4.7: Column dialog

The column dialog triggers the design action

$$
\begin{aligned}
a_{D_{51}} \quad = \quad & \langle \mathsf{makeColumn}, \\
& \{\mathsf{StateConsistentp}()\}, \\
& \{\mathsf{Storey}(\mathsf{x}_1 \rightarrow \mathsf{G6}), \mathsf{setPoint}(\mathsf{P}_3 = (15.0, 0.0)), \\
& \ \mathsf{chooseConcrete}(\mathsf{RC} \rightarrow \mathsf{C25}), \\
& \ \mathsf{selectColumnType}(\mathsf{C1} \rightarrow \mathsf{Rectangular}), \\
& \ \mathsf{chooseLoadElements}(\mathsf{lst}_1 \rightarrow [\mathsf{SL6}]), \\
& \ \mathsf{chooseSupportElements}(\mathsf{lst}_2 \rightarrow \mathsf{nil}), \\
& \ \mathsf{setLength}_m(\mathsf{l}_1 \rightarrow 3.0), \mathsf{setBreadth}_m(\mathsf{b}_1 \rightarrow 0.15), \\
& \ \mathsf{setHeight}_m(\mathsf{h}_2 \rightarrow 0.3), \\
& \ \mathsf{setKo}_{z_m}(\mathsf{z}_3 \rightarrow 19.825), \mathsf{setKo}_{z_m}(\mathsf{z}_4 \rightarrow 22.825)\}, \\
& \{\mathsf{RectangularColumn}(\mathsf{C1}), \\
& \ \mathsf{hasPart}(\mathsf{G6}, \mathsf{C1}), \mathsf{hasPoint}(\mathsf{C1}, \mathsf{P}_3), \\
& \ \mathsf{hasCoordinate}(\mathsf{CO1}, \mathsf{z}_3), \mathsf{hasCoordinate}(\mathsf{CO1}, \mathsf{z}_4), \\
& \ \mathsf{hasMaterial}(\mathsf{C1}, \mathsf{C25}), \mathsf{Supports}(\mathsf{C1}, \mathsf{SL6}), \\
& \ \mathsf{Length}_m(\mathsf{C1}, \mathsf{l}_1), , \mathsf{Breadth}_m(\mathsf{C1}, \mathsf{b}_1), \mathsf{Heigth}_m(\mathsf{C1}, \mathsf{h}_2)\}\rangle.
\end{aligned}
$$

After the user has entered the column, the column is tested for consistency on the definitorial knowledge about rectangular columns.

$$\begin{aligned}\mathsf{RectangularColumn} \;=\; & \mathsf{StructuralElement} \sqcap \\ & \exists \mathsf{Normalforce}_{kN}, \mathsf{Capacity}_{kN}.\leq \sqcap \\ & \exists \mathsf{Height}_m.\geq_{0.2} \sqcap \exists \mathsf{Breadth}_m.\geq_{0.2} \sqcap \\ & \exists \mathsf{ReinforcementRatio}.\leq_{[0.03,0.09]} \sqcap \\ & \exists \mathsf{Height}_m, \mathsf{Breadth}_m.\leq_{4\cdot \mathsf{Breadth}_m} \sqcap \\ & \exists \mathsf{Length}_m, \mathsf{Height}_m.\geq_{6\cdot \mathsf{Height}_m} \\ & \exists \mathsf{Slenderness}.\leq_{[\frac{20}{\sqrt{\nu_{Ed}}},25]}\end{aligned}$$

The structural column C1 is represented in the new ABox by the set of assertions in the add list of $a_{D_{51}}$. Because the column contradicts the minimum gross sectional requirement stipulated in the code the user is prompted to revise the column, immediately.

The modify design operator $a_{D_{52}}$ becomes applicable and the other design buttons are blocked. After the user has changed the contradicting value, he can continue with his design. The system deletes first the contradicting assertions of the delete list and adds thereafter the new ones of the add list. Only the modified assertions are given.

$$\begin{aligned}a_{D_{52}} \;=\; & \langle \mathsf{makeColumn}, \\ & \{\mathsf{RectangularColumn}(\mathsf{C1})\}, \\ & \{\mathsf{setHeight}_m(\mathsf{h}_2 \rightarrow 0.3), \\ & \ \mathsf{setBreadth}_m(\mathsf{b}_2 \rightarrow 0.3)\}, \\ & \{\mathsf{Breadth}_m(\mathsf{C1}, \mathsf{b}_1), \\ & \ \mathsf{Heigth}_m(\mathsf{C1}, \mathsf{h}_2)\} \\ & \{\mathsf{Breadth}_m(\mathsf{C1}, \mathsf{b}_2), \\ & \ \mathsf{Heigth}_m(\mathsf{C1}, \mathsf{h}_3)\rangle.\end{aligned}$$

In this way, the engineer creates the storey columns, which he places preferably in the functional grid points. He uses a *copy* operator $a_{D_{70}}$ to copy the columns. All values are retained except of the identifier and the location. They support the flat slab and determine the span length in both directions. The user finishes the column system at the top storey by an action $a_{D_{60}}$ that adds the assertion $\mathsf{ColumnSystem}(\mathsf{G6}, \mathsf{CS1})$ to the current ABox. The system state is now given by $\mathcal{S}_{41} = (\mathcal{A}_{33}, t_6)$ which resulted from the action sequence $\mathcal{P}_5 = [a_{D_{60}}, a_{D_{70}}, \ldots, a_{T_1}, \mathcal{P}_4]$.

After the user has placed the vertical elements, he sizes the flat slab with $\mathsf{l} = 5.0$ in the subtask **sizeFlatSlab** with the design action

$$\begin{aligned} a_{D_{16}} \;=\; & \langle \mathsf{sizeFlatSlab}, \\ & \{\mathsf{StateConsistentp}(), \mathsf{FlatSlab}(\mathsf{SL6}), \mathsf{SpanLength}_{max}(\mathsf{SL6}, \mathsf{l})\}, \\ & \{\mathsf{calculateDepth}_m(\mathsf{d}_1 = 0.9 \cdot 5.0/35), \\ & \ \mathsf{calculateHeight}_m(\mathsf{h}_4 = \mathsf{d}_6 + 0.04), \\ & \ \mathsf{calculateSelfWeight}_{kN/m^2}(\mathsf{g}_{S_6} = 25 \cdot \mathsf{h}_6)\}, \\ & \{\mathsf{Depth}_m(\mathsf{SL6}, \mathsf{d}_1), \mathsf{Heigth}_m(\mathsf{SL6}, \mathsf{h}_4), \\ & \ \mathsf{supports}(\mathsf{SL6}, \mathsf{g}_{S_6}), \\ & \ \mathsf{SelfWeight}_{kN/m^2}(\mathsf{SL6}, \mathsf{g}_{C_1}), \\ & \ \mathsf{Designable}(\mathsf{SL6})\}\rangle. \end{aligned}$$

It calculates the required slab depth by a limitation for deflection from the DIN 1045-1 for which the total slab height can be approximately assigned. The user can not yet copy aggregated objects like a storey system. Therefore, he has to design in a similar way each storey system from scratch in the following up subtasks of t_5 for which the last finish task adds the assertion $\mathsf{VerticalSystem}(\mathsf{VS})$ and $\mathsf{hasPart}(\mathsf{CONCEPT}, \mathsf{VS})$. For the next subtask **sizeStructuralElements** in

$$t_7 = [\ldots, \textbf{sizeStructuralElements}, \mathsf{makeFoundation}, \mathsf{calculateLability}],$$

a set of design operators are applicable depending on the type of elements like columns, beams, and footings, in the structural concept. Before a structural element can be sized, the internal forces of the specific elements must be derived by tributary areas. For an internal column at the ground level, the system determines $\mathsf{NormalForce}_{kN}(\mathsf{C170}, \mathsf{N}_{Ed})$ with $\mathsf{N}_{Ed} = 1350$. In the example, the user can choose from the operator set $A_{app} = \{a_{D_{52}}, a_{D_{73}}, a_{D_{74}}\}$. Consider the operator for sizing a column

$$\begin{aligned} a_{D_{52}} \;=\; & \langle \mathsf{makeColumn}, \\ & \{\mathsf{RectangularColumn}(?\mathsf{x}_1), \mathsf{NormalForce}_{kN}(?\mathsf{x}_1, ?\mathsf{x}_2), \\ & \ \mathsf{hasConcreteStrength}(?\mathsf{x}_1, ?\mathsf{x}_3), \mathsf{hasSteelStrength}(?\mathsf{x}_1, ?\mathsf{x}_4), \\ & \ \mathsf{ConcreteGrossSection}(?\mathsf{x}_1, ?\mathsf{x}_5)\}, \\ & \{\mathsf{calculateReinforcement}_{cm^2}(?\mathsf{x}_6 = (?\mathsf{x}_2 - ?\mathsf{x}_3 \cdot ?\mathsf{x}_5)?\mathsf{x}_4)\}, \\ & \{\ \mathsf{Capacity}_{kN}(?\mathsf{x}_1, ?\mathsf{x}_2), \mathsf{Designable}(?\mathsf{x}_1)\}\rangle. \end{aligned}$$

It calculates the required longitudinal reinforcement for a given gross section. In the next two steps, the engineer sizes the footings and thereby finishes the conceptual design task what he indicates by a corresponding finish action.

The subtask **makeFoundation** in t_7 includes the placement of footings for the foundation and their sizing. In this example, it is not problematic due to the allowed high soil pressure and is skipped in the example. Basically, the user can design *line* and *point footings* under the load-bearing elements by the operators $a_{D_{64}}, a_{D_{65}}$ and size them according to well-known rules of thumb from the DIN 1045-1 by operators $a_{D_{66}}, a_{D_{67}}$. The modal dialogs of the point and line footings are similar to the dialogs of the other elements.

Now, under the next subtask **calculateLability** in t_7, only one operator is applicable to *calculate the lability numbers* α_1 and α_2 in both building directions according to the DIN 1045-1. The calculations are carried out in the binding list of the operator.

$$
\begin{aligned}
a_{T_2} \;=\; & \langle \mathsf{calculateLability}, \\
& \{\mathsf{StateConsistentp}(), \mathsf{hasStabilitySystem}(\mathsf{CONCEPT}, \mathsf{LS})\}, \\
& \{\mathsf{calculateLabilityNumber}(\alpha_1 = \frac{1}{\mathsf{BuildingHeight}_m} \cdot \sqrt{\frac{\mathsf{E}_C \cdot \mathsf{I}_{C_x}}{\mathsf{F}_{Ed}}}) \\
& \quad \mathsf{calculateLabilityNumber}(\alpha_2 = \frac{1}{\mathsf{BuildingHeight}_m} \cdot \sqrt{\frac{\mathsf{E}_C \cdot \mathsf{I}_{C_y}}{\mathsf{F}_{Ed}}})\}, \\
& \{\mathsf{hasValue}(\mathsf{LS}, \alpha_1), \mathsf{hasValue}(\mathsf{LS}, \alpha_2), \mathsf{stable}(\mathsf{LS})\}\rangle
\end{aligned}
$$

The system calculates the sum of the flexural rigidities and the vertical forces on the structure in the background. If the values of α_1 or α_2 are smaller than $1/(0.2 + 0.1 \cdot \mathsf{n})$ for $\mathsf{n} \leq 3$ or $1/0.6$ for $\mathsf{n} \geq 4$, where n is the storey number, the system is sway and even additional shear walls or a more refined analysis are needed. In this example, the structure is non-sway and the user continues with the design.

After the last task of calculating the stability the prototype automatically activates the load control. It computes by testing action a_{T_3} again for all elements of the concept if they are designed and supported. For example, supporting elements of columns are defined by the concept StructuralElement and a pair of the completeness axioms for the concept Column from the CBox.

$$
\begin{aligned}
\mathsf{StructuralElement} &\equiv \mathsf{Column} \sqcup \mathsf{Wall} \sqcup \mathsf{Slab} \sqcup \mathsf{SlabBeam} \\
\mathsf{LoadedColumn} &\equiv \mathsf{Column} \sqcap \exists \mathsf{supports}.(\mathsf{StructuralElement}) \\
\mathsf{DesignedColumn} &\equiv \mathsf{Column} \sqcap \ \mathsf{Designable} \sqcap \exists_{n=1} \mathsf{loads}.(\mathsf{Column} \sqcup \mathsf{Wall})
\end{aligned}
$$

The completeness test returns no incomplete column objects because all loaded columns are also designed and supported. In the case of the whole concept, the set of all pairs of completeness axioms is empty since no elements have incomplete load paths. A common model exists for the complete and consistent ABox $\mathcal{A}$ on the CBox $\mathcal{C}$ and the TBox $\mathcal{T}$. Hence, the final system state $\mathcal{S}_{65} = (\mathcal{A}_{55}, [])$ resulting from the action sequence $\mathcal{P}_6 = [a_{T_3}, a_{T_2}, \ldots, \mathcal{P}_5]$ is a goal state. This sequence constructs first an initial concept $\mathcal{I}_0$ and refines it then into the goal concept $\mathcal{I}_G$ given as follows:

$$
\begin{aligned}
\Delta_{\mathcal{I}_G} &= \Delta_{\mathcal{I}_0} \cup \{\mathsf{VS}, \mathsf{FS}, \mathsf{CS1}, \mathsf{SW1}, \mathsf{SW2}, \mathsf{C1}, \mathsf{FO1}, \ldots\},\\
\mathsf{StructuralObject}^{\mathcal{I}_G} &= \mathsf{Load}^{\mathcal{I}_G} \cup \mathsf{Element}^{\mathcal{I}_G} \text{ with all } x \in \mathsf{Footing}^{\mathcal{I}_G} \text{ minimal},\\
\mathsf{Load}^{\mathcal{I}_G} &= \mathsf{Load}^{\mathcal{I}_0} \cup \{\mathsf{g}_{S_1}, \mathsf{g}_{S_2}\},\\
\mathsf{Element}^{\mathcal{I}_G} &= \mathsf{Element}^{\mathcal{I}_0} \cup \{\mathsf{SW1}, \mathsf{SW2}, \mathsf{C1}, \mathsf{FO1}, \ldots\},\\
\mathsf{Footing}^{\mathcal{I}_G} &= \{\mathsf{FO1}, \ldots, \mathsf{FO31}\},\\
\mathsf{StoreySystem}^{\mathcal{I}_G} &= \mathsf{StoreySystem}^{\mathcal{I}_0},\\
\mathsf{System}^{\mathcal{I}_G} &= \mathsf{System}^{\mathcal{I}_0} \cup \{\mathsf{CO1}, \ldots, \mathsf{VS}, \mathsf{FS}\},\\
\mathsf{Designable}^{\mathcal{I}_G} &= \mathsf{Element}^{\mathcal{I}_G} \text{ and } \mathsf{Stable}^{\mathcal{I}_G} = \{\mathsf{LS}\},\\
\mathsf{Constraint}^{\mathcal{I}_G} &= \mathsf{Constraint}^{\mathcal{I}_0}\\
\mathsf{supports}^{\mathcal{I}_G} &= \mathsf{supports}^{\mathcal{I}_0} \cup \{(\mathsf{C1}, \mathsf{SL6}), (\mathsf{FO1}, \mathsf{C188}), (\mathsf{FO1}, \mathsf{CO1}), \ldots\},\\
\mathsf{hasPart}^{\mathcal{I}_G} &= \mathsf{hasPart}^{\mathcal{I}_0} \cup \{(\mathsf{CONCEPT}, \mathsf{VS}), (\mathsf{VS}, \mathsf{G1}), (\mathsf{G1}, \mathsf{CO1}),\\
&\qquad (\mathsf{CO1}, \mathsf{C1}), (\mathsf{CONCEPT}, \mathsf{FS}), (\mathsf{FS}, \mathsf{FO1}),\\
&\qquad (\mathsf{LS}, \mathsf{SW1}), \ldots\}, \text{ and}\\
\mathsf{hasConstraint}^{\mathcal{I}_G} &= \mathsf{hasConstraint}^{\mathcal{I}_0}.
\end{aligned}
$$

New objects in the domain are the vertical system VS, the foundation system FS, the column systems $\mathsf{CS1} - \mathsf{CS6}$ at each storey, the shear walls $\mathsf{SW1}, \mathsf{SW2}$, the columns $\mathsf{C1}$-$\mathsf{C188}$, and the footings $\mathsf{FO1} - \mathsf{FO31}$. The set of designable elements includes the slabs, the columns, and the lateral elements since they can always be designed. For the heaviest loaded column, the partial functions $\mathit{state\text{-}variable}^{\mathcal{I}_0}$ and $\mathit{parameter}^{\mathcal{I}_0}$ are for example as follows:

$$
\begin{aligned}
\mathsf{NormalForce}^{\mathcal{I}_0}_{kN} &= (\mathsf{C170}, \mathsf{N}_{Ed})\\
\mathsf{Capacity}^{\mathcal{I}_0}_{kN} &= (\mathsf{C170}, \mathsf{N}_{Rd}) \text{ with } \mathsf{N}_{Ed} = 1350 \leq \mathsf{N}_{Rd} = 1360.
\end{aligned}
$$

The concept is depicted in figure 4.8. It encompasses a column system with a 5.0 m $\times$ 5.0 m grid and 0.30 m $\times$ 0.30 m column dimensions, a flat slab system with a height of 18 cm, a stability system with two cores and one shear wall, and a foundation with point and line footings.

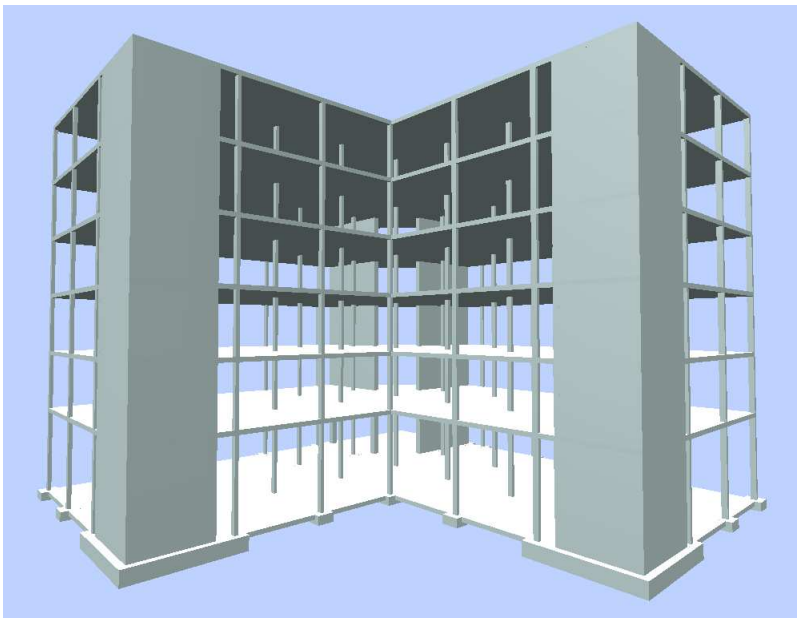


Figure 4.8: Final structural concept

4.3 Use case scenarios

The prototype system has been developed under the assumptions for ordinary buildings described in Section 1.3. They match roughly the requirements for a building classified as ordinary building structure according to DIN 1045-1. Because the acquired knowledge does not cover the full range of design support for ordinary building structures the system's scope of application is given in table 4.1.

The prototype system has been designed for the use of structural engineers who collaborate with the architect in the early phase of conceptual building design. At this point of the design process, only main building elements that steam from the occupancy and room concept of the architect are given. Therefore, I assumed that the user of the system has to design a structural concept with its elements. In practice however, architects often sketch an incomplete structural concept with chosen element types that has to be completed and/or modified by structural engineers. To aid the completion and modification the system has to draw on the same design support as the one developed in this thesis. To avoid entering an incomplete concept a second time an implementation of the Industry Foundation Classes (IFC) data import functionality is planned. IFC is an object-oriented product model for

Requirement type	Limitation
Occupancy	living, office, factory, residence, manufacture
Layout	L-shaped, H-shaped, U-shaped, S-shaped, rectangular, layouts composed of simple geometries
Building height	$h \leq 23$m
Material	reinforced concrete of C16/20-C35/45 classes
Load magnitude	steady, occupancy $q \leq 5\text{kN/m}^2$ and no extraordinary concentrated loads Q
Soil	structural soil like clay, etc.
Spanning length	$l \leq 16$m
Element	column, beam, slab, core, shear wall, deep beam, wall, point footing, line footing
Load	point load, line load, distributed area load, wind, snow
Slab	flat, one-way, two-way, prefabricated planks
Gross section	rectangular, circular only for columns

Tabelle 4.1: Limitation for system applicability

3D building data exchange, which is supported by some major CAD vendors. Since many architects work 2D oriented, the open question remains how to bridge the gap of 2D data generation and 3D data exchange. Other exchange standards will not be supported since they are not object-oriented and can thus not avoid the work for structural engineers to enter the object data. The structural concept with its preliminarily sized members is used in detailed design and formwork drawings. Therefore, an export of element data to design programs as well as an IFC export to drawing programs is a reasonable extensions for the system in order to integrate the developed system fully in the planning process.

Chapter 5

Conclusions

This chapter summarizes the major thesis results and discusses to what extend the research problem defined in Section 1.3 has been solved. I state the contribution of the thesis to the state of knowledge in the research area due to the findings of the thesis. They are compared to other proposals for computer-aided design support of building structures. At last, I point out open questions, which remain to be solved by further development in the research area.

5.1 Summary of Results

The thesis developed a knowledge-based formalism in Chapter 3 to support engineers in conceptual design of building structures. It is based on a model of conceptual design. The model is founded on a vast body of design knowledge as described in Chapter 2. Design knowledge encompasses conceptual knowledge about robust structural concepts and control knowledge about the organization of the design process. Conceptual knowledge from acquired from textbooks is represented in a description logic as described in Subsection 3.3.2, and the control knowledge elicited from structural experts in a hierarchical planning language on top of it, respectively, see Subsection 3.3.5. The model maps the experienced engineers' reasoning about the correctness of the structural concept and the planning of the conceptual design task to computational processes. The result is a description logic based planning algorithm, which was presented in Section 3.3.6. It employs the consistency algorithm of Section 3.3.3 for testing the structural concept's correctness. Implemented into a computer-aided conceptual design system, engineers can reduce error rates of structural concepts and the time required for the design process. The formalism enhances the possible support of such systems as shown in

a detailed example in Chapter 4. Furthermore, the developed formalism allows to encode the conceptual and control knowledge in a declarative way as its representation is independent of its processing by the algorithms. The knowledge is thus versatile employable and independent of the underlying programming language. However, it is still tailored for conceptual design of ordinary building structures and must be first adapted to other purposes.

The formalism should support the engineer as stated in Section 1.3 in: 1. testing the correctness of a structural concept, 2. sequencing subtasks and actions before flexibly solving the subtasks in a bottom-up fashion, and 3. proposing design actions to develop and reinstall the correctness of the concept.

1. Testing the correctness of the structural concept is carried out by the reasoning services of consistency and completeness testing in definition 15 and 16. Consistency testing works on the definitorial part of the conceptual knowledge, which structural concepts always have to satisfy. Completeness testing works on the background part by which structural elements are computed that need additional supporting elements to ensure local load transfer. For consistency testing, definitorial knowledge about structural elements stipulated in the norm DIN 1045-1 is used, which is stored in a TBox. A column was considered as example in Section 4.2, which has to fulfill minimum gross sectional requirements. If the user designs an inconsistent element, which violates the requirements represented in the definitorial part, it causes a direct contradiction and is immediately detected in the formalism. The structural concept remains inconsistent, unless the user revises the chosen gross sectional parameters of the considered column. For completeness testing, background knowledge, stored in a CBox, can be applied to control the load transfer among structural elements at certain points during conceptual design. An example was presented in Section 4.2 for a shear wall. If the engineer activates the control of the vertical load transfer during the design process, the reasoning service computes the set of incomplete elements. It highlights these elements for which additional supporting elements are required or which still have to be sized. It is up to the engineer, whether or not to revise the highlighted elements. The consistency algorithm in Section 3.3.3 implements the reasoning services of consistency and completeness testing. It is capable of dealing with incomplete information without signaling an error, since most load paths are incomplete especially at the beginning and during the design process. The user can decide when to complete or revise load paths for securing overall load transfer. In this way, finding

inconsistent or incomplete structural objects can be intuitively performed in an object-oriented manner.

2. The engineer can plan his design approach by means of the interactive CONED algorithm. It supports a goal-directed and hierarchical approach, which is applied from experienced engineers to design a concept by a forward search decomposition process, see definition 4. The algorithm enforces the engineer to search depth-first for suitable subtask decompositions at different task levels. At the conceptual level, it is combined with some kind of breadth search to assign a reduced set of possible design actions under a current system state. Hence, goal-directed work is supported on the task levels but the engineer can still design the structural concept bottom-up at the conceptual level. On the conceptual level, the engineer can directly manipulate the structural concept and reason about its correctness by means of the conceptual knowledge during the design process. The necessary direct manipulation actions were defined in Section 2.5. They have been presented in the example of Section 4.2. A decomposition of a slab into independent slab portions cannot be performed by the user. Although it is a desirable support feature for conceptual design where the design process proceeds from the abstract to detailed level, it becomes a difficult task for aggregated objects with complex geometries. On the task level, the algorithm draws on experienced practitioners' control knowledge about sequencing tasks in a suitable order. Section 4.2 showed how planning can reduce the number of iteration cycles for engineers and thus limit the time they spend to design a structural concept. This is especially important for novices who lack the notion of planning in their mental models about conceptual design. The planning is founded on appropriate task decompositions of conceptual design, which depend on the initial structural concept at hand. Inexperienced engineers can avoid thereby long backtracking paths that result from inappropriate subtask sequencing.

3. The algorithm 3.4 proposes design actions to correct the developing concept in simple tasks. Planned tasks that have not been performed are hence differently decomposable depending on the current state of the concept. Interleaving subtasks and partial orderings among subtasks in tasklists are not supported. Therefore, some flexibility of design support is lost due to the chosen type of planning. This type of planning support might be to inflexible for experienced practitioners because it enforces a strict subtask sequencing.

As a proof of concept, the formalism has been implemented in a prototypical computer-aided design system, which was presented in Chapter 4 on a trace of the underlying formalism. The prototype used the reasoner package RACER [HM04] to implement the conceptual model with the respective reasoning services and the planning package SHOP [NCLMA04] to provide the functionality for task decomposition planning. The prototype is implemented in LISP and CLOS together with the GUI development environment from FRANZ [Fra04].

Besides the developed formalism, conceptual design of concrete structures has been formalized, see definition 4. It makes the mental model shared by experienced practitioners explicit in terms of required knowledge and reasoning. Design knowledge encompasses so-called conceptual knowledge about structure and load transfer and control knowledge about appropriate sequencing of subtasks and design actions upon practitioners' reasoning operate. On this formalization, conceptual design of concrete building structures can be understood as a goal-directed search process through the design state space for a correct structural concept in a focused manner. The design state space spans over the set of conceivable maybe incomplete and inconsistent structural concepts, which the engineer explores by design actions to build up a complete and consistent one. At the conceptual level, conceptual knowledge serves experienced engineers to detect incorrect concepts. At the task level, their control knowledge selects promising search paths through the very large state space.

5.2 Discussion

The thesis developed a model of conceptual design in the domain of concrete building structures resulting in definition 4. The definition was based on a thorough textbook study, a number of interviews, and design studies conducted at structural design offices. The studies elicited important internal actions, also called reasoning skills, that have only partially or not at all taken into account by other proposals, which have targeted different design support or focused on computational aspects. Therefore, a formulation has been missing considering also internal actions on the mental model, although they have been implicitly applied by experienced engineers.

In addition to that, requirements for interactive knowledge-based design systems have to be considered. As stated in [GRS00], a high degree of interaction and traceability determines the usability of such systems. SHNEIDERMAN [Shn98] understands direct model manipulation as a major usability require-

ment for GUIs of computer-aided design systems, too. WOLTER even describes that the system's acceptance by the engineer strongly depends on this fact [Wol03]. As found out by the testing scenarios for the GUI design in this thesis, the computer-aided design system's traceability corresponds to a close fit between the users' mental models and the system model, especially for the planning aspect. Therefore, proposals for interactive conceptual design assistance should take these requirements into account. The developed prototype system tries to cover these by making direct manipulation actions possible and by introducing an additional GUI component, the task structure dialog, for planning, first proposed by HAUSER in [HS97].

HAUSER remarks about the PRED system in [Hau98] that a fixed sequence of design actions may be too rigorous for providing ergonomic design assistance. He states in [Hau98] that the design process is iterative in nature and is carried out with focus on alternating abstraction levels. This idea led to a cognitive architecture of conceptual designing, which possesses a *strategic*, a *tactical* and a *reactive* level. The strategic level implements the planning, the tactical level the execution of design actions for selecting element alternatives and sizing them, and the reactive level the local modification of element parameters due to geometrical constraints, respectively. However, planning and execution, as well as execution and modification cannot be interleaved in this framework as the levels interact top-down. Therefore, a plan in form of an abstract concept is first completely laid out that determines a fixed sequence of design actions on the tactical level. Design actions construct then a structural concept as specified on the strategic level. A revision of the plan in response to design actions is not possible. After the concept has been developed on the tactical level the reactive level ensures that local element constraints are satisfied. Since conceptual knowledge is not declaratively represented in the conceptual model of the architecture, the engineer cannot design self-contained structural systems that can be tested for correctness [ES01b]. Whether or not a structural system like for example the stability system is correct, affects however the sequencing of subtasks, and hence the planning. GEHRE [Geh99] also points out that the engineer needs a possibility for direct element manipulations by element related dialogs.

The CONFIG design system does not explicitly represent the design process [Gom98]. The engineer's need to plan the design process and implement it in a flexible manner is hence not supported. Elements can be refined along an inheritance hierarchy given on a dialog for user input in the CONFIG system [FRG00]. This includes a number of refinement steps excluding modifications. Since engineers usually perform one or at most two design actions for each building element and work iteratively, the operational scale is too fine

and too strict for effective use in practice [Eis00, Eis01a]. Furthermore, my notion of planning differs from the one proposed by GOMEZ [FRG00] and MAHER [Mah90]. I advocate planning as a sequence of planning, testing, and design actions where the sequence is dynamically constructed on the current system state. The task of establishing a global load transfer is not covered by GOMEZ's and MAHER's work because they do not represent abstract load paths, which are important to set up the structural configuration as described in example 2. Therefore both systems, CONFIG and HIGH-RISE, cannot assist the engineer in testing a structural concept's correctness during design, i.e. at a time when the engineer has only incomplete information about structural systems and elements.

The approaches of BRID [SMM99] and CADREM [KR97] do not contain a system model of the structural concept. It is difficult for the user to conceive the obtained results as pointed out by MILES in [Mil01]. He mentions that engineers face the problem of how to understand the results of the search performed by the system. Therefore, he proposes to display selected two-dimensional hyper-planes described by pairs of selected design variables. Thereby, the user can learn about the design space and the dependencies among the variables. This help is however of restricted use for conceptual design problems, which demand a high degree of interaction and cannot be formulated as numerical optimization problems on a set of algebraic equations. As a consequence, users requested a graphical representation [SMM03].

The reviewed configuration design methods in Section 3.2 work mostly for routine design. Routine design is carried out top-down along a predefined system or product structure [GK99]. In conceptual design, this is not longer possible since the structure has to be determined on a component-based conceptual model. Main systems of a structural concept are designed in a bottom-up fashion whereby the necessary degree of interaction increases among the engineer and the computer-aided design system. Nevertheless ideas from classical configuration design were used to develop the system but had to be adapted to conceptual design of building structures. Furthermore, evaluating possible goal concepts in terms of expected behavior to ensure the correctness of a goal configuration becomes more important for conceptual design. This requirement is mostly not studied in the research area of configuration design.

5.3 Further Work

Systems for computer-aided conceptual structural design are a young research area, which has started in the eighties with the work of MAHER's system HIGH-RISE [Mah84]. Although much research work has been done since then, see for a summary [Mil01], a lot of questions remain open because of the still unprecise problem definition of conceptual structural design as such. Because a problem definition did not exist to use as a starting point for the thesis, the thesis defined first a conceptual design problem, see definition 3. This definition is independent of the computational methods employed to solve it. A major goal of further work should be therefore to develop a precise problem definition as a foundation on which applied computational methods become comparable[1]. However, they should take into account the high demand of interaction for exploration tasks such as conceptual design. The thesis stands for a first step in this direction. More work is necessary to refine the model of conceptual design of definition 4 on a range of thoroughly analyzed design studies, see as example the analysis work about conceptual architectural design of KANNENGIESSER and GERO [KG02].

Further work that arises from open question of the thesis can be roughly distinguished into two categories. The first one concerns the improvement of the developed formalism. The following open questions remain:

- how to represent quantified effects in operators?
- how to support the interleaving of subtasks?
- how to make finished subtasks revisable?
- how to make all actions available even though applicable actions are preferable?
- how to incorporate actions for checking conflicts?

Quantified effects can automatically extend the design of a single element to all elements of its equivalence class. Interleaving of subtasks requires to search in different branches of the decomposition tree and not only from left to right in the search tree. Making subtasks revisable needs a modification of the planning algorithm. If actions for checking conflicts have to be processed, a predicate is necessary to direct the design process to the conflicting points.

[1] As an example of problem formulation work and its continuous refinement see configuration design [MF89, WS97, Bro97].

Knowledge about when and how to tackle these conflicts in the design process has to be acquired by additional design studies.

The second aspect concerns the extension of the represented knowledge in the conceptual design domain and the integration of other knowledge types. Work has to be done to:

- make free composition and decomposition of structural objects possible,
- represent slabs with irregular geometry and enable their decomposition,
- add further design requirements of the DIN 1045-1 that determine when certain support relations hold, and
- automate the calculation of internal forces in the conceptual model or by means of external analysis packages.

I made a number of assumptions about the structural characteristics of the building, these should also be relaxed to extend the design support to a wider range of building types.

References

[AF00] A. Artale and E. Franconi. A Survey of Temporal Extensions of Description Logics. *Annals of Mathematics and Artificial Intelligence*, 30(1-4):171–210, 2000.

[AFGP96] A. Artale, E. Franconi, N. Guarino, and L. Pazzi. Part-Whole Relations in Object-Centred Systems: An Overwiew. *Data and Knowledge Engineering*, 20:347–383, 1996.

[BA97] P. Borst and H. Akkermans. Engineering Ontologies. *International Journal Human-Computer Studies*, 46:365–406, 1997.

[Baa99] F. Baader. Logic-Based Knowledge Representation. In M.J. Wooldridge and M. Veloso, editors, *Artificial Intelligence Today, Recent Trends and Developments*, number 1600 in Lecture Notes in Computer Science, pages 13–41. Springer Verlag, 1999.

[Baa02] F. Baader. Logic-based Knowledge Representation. Chair of Automata Theory, Technical University of Dresden, 2002. Lecture Notes.

[Bas00] P. Basieux. *Die Architektur der Mathematik - Denken in Strukturen.* Rowolth Verlag, 2000.

[Bau01] Normenausschuss Bauwesen. *DIN 1045-1.* DIN Deutsches Institut für Normung e.V., Berlin, 2001.

[BBH94] F. Baader, M. Buchheit, and B. Hollunder. Cardinality Restrictions on Concepts. In *Proceedings of the German AI Conference, KI'94*, volume 861 of *Lecture Notes in Computer Science*, pages 51–62, Saarbrücken (Germany), 1994. Springer–Verlag.

[BC89] D.C. Brown and B. Chandrasekaran. *Design Problem Solving, Knowledge Structures and Control Strategies.* Morgan Kaufmann Publishers, 1989.

[BCM+02] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *Description Logic Handbook*. Cambridge University Press, 2002.

[BD95] E.J. Brien and A.S. Dixon. *Reinforced and Prestressed Concrete Design-The Complete Process*. Longman Scientific and Technical, 1995.

[BG01] B. Bonet and H. Geffner. Planning and Control in Artificial Intelligence: A Unifying Perspective. *Applied Intelligence*, 14(3):237–252, 2001.

[BH91] F. Baader and P. Hanschke. A Scheme for Integrating Concrete Domains into Concept Languages. DFKI research report RR-91-10, Deutsches Forschungszentrum für Künstliche Intelligenz, Kaiserslautern, 1991.

[BHS93] W. Bibel, S. Hölldobler, and T. Schaub. *Wissensrepräsentation und Inferenz*. Vieweg Verlag, 1993.

[BKN95] M. Buchheit, R. Klein, and W. Nutt. Constructive Problem Solving: A Model Construction Approach towards Configuration. DFKI technical memo TM-95-01, Deutsches Forschungszentrum für Künstliche Intelligenz Saarbrücken, 1995.

[BLSW02] F. Baader, C. Lutz, H. Sturm, and F. Wolter. Fusions of description logics and abstract description systems. *Journal of Artificial Intelligence Research (JAIR)*, 16:1–58, 2002.

[Bor96] A. Borgida. On the Relative Expressive Power of Description Logics and Predicate Calculus. *Journal of Artificial Intelligence*, 82:353–367, 1996.

[Bro97] D.C. Brown. Defining Configuring. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 12(4):301–305, 1997.

[BS01] F. Baader and U. Sattler. An Overview of Tableau Algorithms for Description Logics. *Studia Logica*, 69:5–40, 2001.

[Bur98] J.E. Burge. Knowledge Elicitation for Design Task Sequencing Knowledge. Master's thesis, School of Computer Science, Worchester Polytechnic Institute, 1998.

[CCD96] N. Cross, H. Christianaas, and K. Dorst. *Analyzing Design Activity*. John Wiley & Sons, 1996.

[Cha90] B. Chandrasekaran. Design Problem Solving: A Task Analysis. *AI Magazine*, 11(4):59–71, 1990.

[Cha03] B. Chandrasekaran. Understanding Control at the Knowledge Level. at http://www.cis.ohio-state.edu/chandra/, last visited 4.9.2003.

[CJ93] B. Chandrasekaran and T. R. Johnson. Generic Tasks and Task Structures: History, Critique and new Directions. In J.-M. David, J.-P. Krivine, and R. Simmons, editors, *Second Generation Expert Systems*, pages 232–272. Berlin: Springer-Verlag, 1993.

[CJ00] B. Chandrasekaran and John R. Josephson. Function in Device Representation. *Engineering with Computers*, 16:162–177, 2000. Special Issue on Computer Aided Engineering.

[CJS92] B. Chandrasekaran, T. Johnson, and J. W. Smith. Task Structure Analysis for Knowledge Modeling. *Communications of the ACM*, 33(9):124–136, 1992.

[Cla97] W.J. Clancey. *Situated Cognition*. Cambridge University Press, Cambridge, 1997.

[CRR+90] R.D. Coyne, M.A. Rosenman, A.D. Radford, M. Balachandran, and J.S. Gero. *Knowledge-Based Design Systems*. Addison-Wesley, 1990.

[Cun92] R. Cunis. *Das 3-stufige Framerepräsentationskonzept - eine mehrdimensionale Basis zur Entwicklung von Expertensystemkernen*. infix Verlag, 1992.

[Dav99] R. Davis. Knowledge-Based-Systems. In R. A. Wilson and F. Keil, editors, *The MIT Encyclopedia of Cognitive Sciences*. Bradford, 1999.

[Dek00] K. J. Dekker. Conceptual Design of Concrete Structures. Master's thesis, Chalmers University Gothenburg, Faculty of Civil Engineering, 2000.

[DINR96] G. DeGiacomo, L. Iocchi, D. Nardi, and R. Rosati. Moving a Robot the KRR Approach at work. In *Proc. of the 5th Int. Conf.*

on the Principles of Knowledge Representation and Reasoning (KR'96), pages 198–209. Morgan Kaufmann Publishers, 1996.

[DLN+93] F.M. Donini, M. Lenzerini, D. Nardi, W. Nutt, and A. Scharf. Queries, Rules and Definitions as Epistemic Statements in Concept Languages. DFKI research report RR-93-40, Deutsches Forschungszentrum für Künstliche Intelligenz, Kaiserslautern, 1993.

[dP01] Bundesvereinigung der Prüfingenieure. Anforderungen an das Aufstellen EDV-unterstützter Standsicherheitsnachweise. *Der Prüfingnieur*, pages 49–54, April 2001.

[dP03] Bundesvereinigung der Prüfingenieure. Anforderungen an das Aufstellen EDV-unterstützter Standsicherheitsnachweise - gewöhnliche Hochbauten. *Der Prüfingnieur*, pages 79–82, April 2003.

[Dör01] D. Dörner. *Die Logik des Mißlingens - Strategisches Denken in komplexen Situationen.* Rowolth Verlag, 2001.

[EFT96] H.-D. Ebbinghaus, J. Flum, and W. Thomas. *Einführung in die mathematische Logik.* Spektrum Verlag, 1996.

[EHN94] Kutluhan Erol, James A. Hendler, and Dana S. Nau. UMCP: A Sound and Complete Procedure for Hierarchical Task-Network Planning. In *Artificial Intelligence Planning Systems*, pages 249–254, 1994.

[EHN95] K. Erol, J. Hendler, and D. Nau. Complexity Results for HTN Planning. Technical Report UMIACS-TR-94-32, Computer Science Department, Institute for Systems Research, and Institute for Advanced Computer Studies, University of Maryland, 1995.

[Eis00] M. Eisfeld. Conceptual Computerized Structural Design - Methods for Documentation, Generation and Evaluation. Master's thesis, Chalmers University Gothenburg, Faculty of Civil Engineering, 2000.

[Eis01a] K. Eisenblätter. Wissensbasierter Tragwerksentwurf - Entwicklung einer Task Structure mit Hilfe von Arbeitsstudien

für den Bereich des Konstruktiven Hochbaus in einer HTN-Planungsumgebung. Master's thesis, Technical University of Dresden, Faculty of Civil Engineering, 2001.

[Eis01b] W. Eisfeld. Defficiencies of conceptual structural design in practise. not published, 2001. personal communication.

[Eng98] B. Engström. Concrete Structures - Advanced Course. Division of Concrete Structures, Chalmers University of Technology, 1998. Lecture Notes.

[Eng99] H. Engel. *Tragsysteme*. Gerd Hatje Verlag, 1999.

[ENS95] K. Erol, D. Nau, and V.S. Subrahmanian. Complexity, Decidability and Undecidability Results for Domain-Independent Planning. *Artificial Intelligence*, 76(1-2):75–88, July 1995.

[Ero95] K. Erol. *Hierarchical Task Network Planning: Formalization, Analysis, and Implementation*. PhD thesis, University of Maryland, 1995.

[ES01a] M. Eisfeld and R. J. Scherer. Kontrollwissen für den konstruktiven Entwurfsprozess vom Tragwerksentwurf bis zur Vorbemessung. In R. Romberg and M. Schulz, editors, *Forum Bauinformatik 2001*, pages 135–145. VDI-Verlag, 2001.

[ES01b] M. Eisfeld and R. J. Scherer. Search-Control Knowledge for the Interoperability Problem between Conceptual and Preliminary Structural Design. In B.H.W. Topping, editor, *6th International Conference on the Application of AI to Structural Engineering*, conference notes. Civil Comp Press, 2001.

[ES02a] M. Eisfeld and R. J. Scherer. Conceptual Parametric Design with Expressive Description Logic. In M. Schnellenbach-Held and H. Denk, editors, *Proceedings of the 9th International EG-ICE Workshop*, Advances in Intelligent Computing in Engineering, pages 112–123, Düsseldorf, 2002. VDI-Verlag.

[ES02b] M. Eisfeld and R. J. Scherer. Wissensbasierte Entwurfsunterstützung für die Konzipierung von Tragwerken in der frühen Entwurfsphase. Technical report, Institute of Building Informatics, Technical University Dresden, 2002.

[FH90] E. Frieling and I. Hilbig. Informationstechniken in der Kontruktion. In C.G. Hoyos and B. Zimolong, editors, *Ingenieurpsychologie*, 12, pages 365–395. Göttigen: Hogrefe-Verlag, 1990.

[For88] K. Forbus. Intelligent Computer-Aided Engineering. *AI Magazine*, 9(3):23–36, 1988.

[Fra66] G. Franz. *Konstruktionslehre des Stahlbetons - Grundlagen und Bauelemente*. Springer Verlag, 1966.

[Fra69] G. Franz. *Konstruktionslehre des Stahlbetons - Tragwerke*. Springer Verlag, 1969.

[Fra81] D. Fraser. *Conceptual Design and Preliminary Analysis of Structures*. Pitman Publishing Limited, 1981.

[Fra04] Franz. Franz Allegro LISP. at http://www.franz.com, last visited 2.2.2004.

[Fre99] M. French. *Conceptual Design*. Springer Verlag, 1999.

[FRG00] S.J. Fenves, H. Rivard, and N. Gomez. SEED-CONF: A Tool for Conceptual Structural Design in a Collaborative Building Design Environment. *Artificial Intelligence in Engineering*, 14(3):233–247, 2000.

[Fri93] G. Fricke. Kontruieren als flexibler Problemlöseprozess. In *Fortschrittsberichte, Reihe 1*, 227. VDI-Verlag, 1993.

[FS99] G. Friedrich and M. Stumptner. Consistency-based Configuration. In *Technical Report WS-99-05 of AAAI'99 Workshop on Configuration*, pages 35–40. AAAI Press, 1999.

[Geh99] A. Gehre. Modellierung von Expertenwissen für den wissensbasierten Tragwerksentwurf. Master's thesis, Technical University of Dresden, Faculty of Civil Engineering, 1999.

[Ger90] J. Gero. Design Prototypes : A Knowledge Representation Schema for Design. *AI Magazine*, 11(4):26–36, 1990.

[Ger98a] J. Gero. Conceptual Design as a Sequence of Situated Acts. In I. Smith, editor, *Artificial Intelligence in Structural Engineering*, number 1454 in Lecture Notes in Artificial Intelligence, pages 165–177. Springer Berlin, 1998.

[Ger98b] J.S. Gero. Towards a Model of Designing which includes its Situatedness. In H. Grabowski, S. Rude, and G. Grein, editors, *Unviversal Design Theory*. Shaker Verlag, Aachen, 1998.

[GGF94] R. Ganeshan, J.H. Garrett, and S. Finger. A Framework for Representing Design Intent. *Journal of Design Studies*, 15(1):59–84, 1994.

[GK99] A. Günter and C. Kühn. Knowledge-Based Configuration-Survey and Future Trends. In F. Puppe, editor, *Expertensysteme '99*, Lecture Notes. Springer, 1999.

[GK02] J. Gero and U. Kannengiesser. The Situated F-B-S Framework. In J. Gero, editor, *Artificial Intelligence in Design'02*, pages 89–104, Dordrecht, 2002. Kluwer.

[Gün89] A. Günter. *Flexible Kontrolle in Expertensystemen zur Planung und Konfigurierung in technischen Domänen*. infix Verlag, 1989.

[GNT03] M. Ghallab, D. Nau, and P. Traverso. *Automated Task Planning*. Oxford University Press, 2003.

[Goe94] V. Goel. A Comparison of Design and Nondesign Problem Spaces. *Artificial Intelligence in Engineering*, 9:53–72, 1994.

[Gom98] N. Gomez. *Conceptual Structural Design Through Knowledge Hierarchies*. PhD thesis, Civil and Environment Engineering, Carnegie Mellon University, 1998.

[GP89] V. Goel and P. Pirolli. Motivating the Notion of Generic Design with Information Processing Theory: The Design Problem Space. *AI Magazine*, 10:18–36, 1989.

[GRS00] G. Görz, C.-R. Rollinger, and J. Schneeberger. *Handbuch der Künstlichen Intelligenz*. Oldenbourg Verlag, 2000.

[GT01] J.S. Gero and H.H. Tang. Differences between Retrospective and Concurrent Potocols in Revealing the Process-Oriented Aspects of the Design Process. *Design Studies*, 21(3):283–295, 2001.

[Hac00] W. Hacker. Konstruktives Entwickeln als Tätigkeit - Versuch einer Reinterpretation des Entwurfsdenkens. Jahresforschungsbericht 76, Institut für Allgemeine Psychologie, TU Dresden, Januar 2000.

[Hau98] M. Hauser. *Tragwerksplanung und Künstliche Intelligenz*. PhD thesis, Technical University of Dresden, 1998.

[HB85] E. Hampe and O. Büttner. *Bauwerk - Tragwerk - Tragstruktur, Band 2*. Ernst & Sohn Verlag, 1985.

[Häg93] S. Hägglund. Introducing Expert Critiquing Systems. *The Knowldge Engineering Review*, 8(4):281–284, 1993.

[HL90] D. Hartmann and K. Lehner. *Technische Expertensysteme*. Springer Verlag, 1990.

[Höl01] S. Hölldobler. Computational Logic. Institute of Artificial Intelligence, University of Dresden, 2001. Lecture Notes.

[HM01] V. Haarslev and R. Möller. Description of the RACER System and its Applications. In *Proc. of International Workshop on Description Logic*, 2001.

[HM04] V. Haarslev and R Möller. RACER: Renamed ABox and Concept Expression Reasoner. at http://www.sts.tu-harburg.de/moeller/racer/, last visited 2.2.2004.

[HMW00] V. Haarslev, R. Möller, and M. Wessel. The Description Logic ALCNHR+ extended with Concrete Domains: Revised Version. KOGS Memo FBI-HH-M-290/00, Fachbereich Informatik der Universität Hamburg, 2000.

[Hol86] A. Holgate. *The Art in Structural Design*. Clarendon Press, 1986.

[HS97] M. Hauser and R.J. Scherer. Application of Intelligent CAD Paradigms to Preliminary Structural Design. *Artificial Intelligence in Engineering*, 11:217–229, 1997.

[HST00] I. Horrocks, U. Sattler, and S. Tobies. Reasoning with Individuals for the Description Logic SHIQ. In David MacAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction (CADE-17)*, number 1831 in Lecture Notes in Computer Science, Germany, 2000. Springer Verlag.

[HU94] J.E. Hopcroft and J.D. Ullman. *Einführung in die Automatentheorie, formale Sprachen, und Komplexitätstheorie.* Addison-Wesley, 1994.

[Hub82] V. Hubka. *Principles of Engineering Design.* Boston: Butterwoth Scientific Verlag, 1982.

[HWG00] O. Hollmann, Th. Wagner, and A. Günter. ENGCON: A Flexible Domain-Independent Configuration Engine. In *Proceedings of the ECAI-Workshop Configuration, ECAI-2000*, 2000.

[Inf97] Gesellschaft Informatik. Ergonomische Gestaltung der Benutzungsschnittstellen von CAD-systemen. *Informatik Spektrum*, 20(4):6–74, 1997.

[IS94] Y. Iwasaki and H. A. Simon. Causality and Model Abstraction. *Artificial Intelligence*, 67(1):143–194, 1994.

[Iwa97] Y. Iwasaki. Qualitative Reasoning and the Sciences of Design. *IEEE Expert: Intelligence Systems*, 12(3):2–4, 1997.

[Joh02] J. Johsephon. Thinking Like a Machine - Knowledge System Approach to Artificial Intelligence. available at http://cis.ohio-state.edu/lair/main/, last visited 5.10.2002.

[Kam97] S. Kambhampati. Refinement Planning as a Unifying Framework for Plan Synthesis. *AI Magazine*, 18(2):67–97, 1997.

[KB86] J. De Kleer and J.S. Brown. Theories of Causal Ordering. *Artificial Intelligence*, 29:33–61, 1986.

[KG02] M. Kavalaki and J.S. Gero. The Structure of Concurrent Cognitive Actions: A Case Study of Novice and Expert Designers. *Design Studies*, 23(1):25–40, 2002.

[KMN02] B. Kraft, O. Meyer, and M. Nagl. Graph Technology Support for Conceptual Design. In M. Schnellenbach-Held and H. Denk, editors, *Advances in Intelligent Computing in Engineering, 9th International EG-ICE Workshop*, pages 1–35, Düsseldorf, 2002. VDI-Verlag.

[KR97] B Kumar and B Raphael. CADREM: A Case-Based System for Conceptual Structural Design. *Engineering with Computers*, 13:153–164, 1997.

[Kre02] Th. Krebs. Erkennen von Benutzerintentionen im inkrementellen Konfigurationsverlauf. Master's thesis, Bereich Intelligente Systeme, TZI, Universität Bremen, 2002. Diplom thesis.

[KW95] W. Krätzig and U. Wittek. *Tragwerke 1 - Theorie und Berechnungsmethoden statisch bestimmter Stabtragwerke*. Springer Verlag, 1995.

[LB87] H. J. Levesque and R. J. Brachman. Expressiveness and Tractability in Knowledge Representation and Reasoning. *Computational Intelligence Journal*, 3:78–93, 1987.

[Leo77] F. Leonhardt. *Vorlesungen über den Massivbau*. Springer Verlag, 1977.

[LPB98] H. Lowe, M. Pechoucek, and A. Bundy. Proof Planning for Maintainable Configuration Systems. *Artificial Intelligence in Engineering, Design, Analysis and Manufacturing*, 12:345–356, 1998.

[LS02] C. Lutz and U. Sattler. A Proposal for Describing Services with DLs. In *Proceedings of the 2002 International Workshop on Description Logics*, 2002.

[Lut02] C. Lutz. Description Logics with Concrete Domains-A survey. In *Advances in Modal Logics Volume 4*. World Scientific Publishing Co. Pte. Ltd., 2002.

[Mah84] M. L. Maher. *HI-RISE: A Knowledge-Based Expert System for the Preliminary Structural Design of High Rise Buildings*. PhD thesis, Carnegie Mellon University, 1984.

[Mah90] M. L. Maher. Process Models for Design Synthesis. *AI Magazine*, 11(4):49–58, 1990.

[Mah91] M.L. Maher. Expert Systems for Structural Design. In D. T. Pham, editor, *Expert systems in engineering*. Springer Verlag, 1991.

[McC00] J. McCarthy. What is Artificial Intelligence. at http://www-formal.stanford.edu/jmc/, 2000. last visited 10.9.2000.

[Mer02] L. Merö. *Die Grenzen der Vernunft, Kognition, Intuition und komplexes Denken*. Rowohlt Verlag, 2002.

[MF89] S. Mittal and R. Frayman. Towards a Generic Model of Configuration Tasks. In *International Joint Conference on Artificial Intelligence*, volume 2, pages 1395–1401, 1989.

[MG96a] M.L. Maher and A. Gomez. The Adaptation of Structural System Designs Using Genetic Algorithms. In *Proceedings of the International Conference on Information Technology in Civil and Structural Engineering Design-Taking Stock and Future Directions*, 1996.

[MG96b] M.L. Maher and A. Gomez. Developing Case-Based Reasoning for Structural Design. *IEEE Expert*, 11(3):42–52, 1996.

[MH69] J. McCarthy and P.J. Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. *Machine Intelligence*, 4:463–502, 1969.

[Mil01] J. C. Miles. Computer Aided Conceptual Design: A Review. In *6th International Conference on the Application of AI to Structural Engineering*, conference notes, pages 107–136. Civil Comp Press, 2001.

[MM94] J. Miles and C. P. Moore. *Practical Knowledge-Based Systems in Conceptual Design*. Springer Verlag, 1994.

[Moo97] C.J. Moore. Decision Support for Conceptual Bridge Design. *Artificial Intelligence in Engineering*, 11(3):259–272, 1997.

[Mos85] J. Mostow. Towards Better Models Of The Design Process. *AI Magazine*, 6(1):44–57, 1985.

[MW98] D. McGuinness and J. Wright. An Industrial Strength Description Logic-based Configurator Platform. *IEEE Intelligent Systems*, 13(4):69–77, July 1998.

[Mye96] K.L. Myers. Strategic Advice for Hierarchical Planners. In Luigia Carlucci Aiello, Jon Doyle, and Stuart Shapiro, editors, *KR'96: Principles of Knowledge Representation and Reasoning*, pages 112–123. Morgan Kaufmann, San Francisco, California, 1996.

[Nav91] D. Navinchandra. *Exploration and Innovation in Design*. Springer Verlag, 1991.

[NCLMA99] D. Nau, Y. Cao, A. Lotem, and H. Munoz-Avila. SHOP: Simple Hierarchical Ordered Planner. UMIACS-TR-9904 CS-TR-3981, Department of Computer Science, and Institute for Systems Research, University of Maryland, 1999.

[NCLMA01] D. Nau, Y. Cao, A. Lotem, and H. Muñoz-Avila. The SHOP Planning System. *AI Magazine*, 22(3):64–91, 2001.

[NCLMA04] D. Nau, Y. Cao, A. Lotem, and H. Muñoz-Avila. SHOP project. at http://www.cs.umd.edu/projects/shop/, last visited 2.2.2004.

[Neb90] B. Nebel. *Reasoning and Revision in Hybrid Representation Systems*. Lecture Notes in Artificial Intelligence 422. Springer-Verlag, 1990.

[Neb96] B. Nebel. Artificial Intelligence: A Computational Perspective. In G. Brewka, editor, *Principles of Knowledge Representation*, pages 237–266. CSLI Publications, 1996.

[Neb00] B. Nebel. On the Expressive Power of Planning Formalisms: Conditional Effects and Boolean Preconditions in the STRIPS Formalism. In J.Minker, editor, *Logic-Based Artificial Intelligence*. Kluwer, Dordrecht, 2000.

[Neb01a] B. Nebel. Knowledge Representation and Reasoning. Institute of Artificial Intelligence, University of Freiburg, 2001. Lecture Notes.

[Neb01b] B. Nebel. Logics for Knowledge Representation. In N. J. Smelser and P. B. Baltes, editors, *International Encyclopedia of the Social and Behavioral Sciences*. Kluwer, Dordrecht, 2001.

[New82] A. Newell. The Knowledge Level. *Artificial Intelligence*, 18:87–127, 1982.

[New90] A. Newell. *Unified Theories of Cognition*. Harvard University Press, 1990.

[NG89] N.J. Nilsson and M.R. Genesereth. *Logische Grundlagen der Künstlichen Intelligenz*. Vieweg Verlag, 1989.

[Nie90] J. Nielsen. Big Paybacks from 'discount' Usability Engineering. *IEEE Software*, 7(3):107–108, 1990.

[Nie94] J. Nielsen. Estimating the Number of Subjects needed for a Thinking Aloud Test. *International Journal of Human Computer Studies*, 41:385–397, 1994.

[NS72] A. Newell and H. A. Simon. *Human Problem Solving*. Prentice-Hall, 1972.

[NSE98] D. Nau, S. J. J. Smith, and K. Erol. Control Strategies in HTN Planning: Theory Versus Practice. In *AAAI-98/IAAI-98 Proceedings*, pages 1127–1133, 1998.

[O'S02] B. O'Sullivan. Interactive Constraint-Aided Conceptual Design. *Journal of Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 16(4), 2002.

[PB96] G. Pahl and W. Beitz. *Engineering Design - A Systematic Approach*. London: Springer Verlag, 1996.

[PMG98] D. Poole, A. Machworth, and R. Goebel. *Computational Intelligence - A Logical Approach*. Oxford University Press, 1998.

[Pup91] F. Puppe. *Einführung in Expertensysteme*. Springer Verlag, 1991.

[Rap95] B. Raphael. *Reconstructive Memory in Design Problem Solving*. PhD thesis, University of Strathclyde, 1995.

[Ret94] M. Rettig. Protyping for Tiny Fingers. *Communication of the ACM*, 37(4):21–27, 1994.

[Rin03] J. Rintanen. Introduction to Automatic Task Planning. Institute of Artificial Intelligence, University of Freiburg, 2003. Lecture Notes.

[RN95] S.J. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[RS03] B. Raphael and I. Smith. *Fundamentals of Computer Aided Engineering*. John Wiley and Sons, 2003.

[Sac80] E. D. Sacerdoti. Problem Solving Tactics. *AI Magazine*, 2(1):7–15, 1980.

[Sat03] U. Sattler. Description Logics for Ontologies. In *Proc. of the International Conference on Conceptual Structures (ICCS 2003)*, LNAI, 2003. To appear.

[Sch80] D. Schodek. *Structures*. Prentice-Hall, 1980.

[Sch83] D.A. Schön. *The Reflective Practitioner, How Professionals Think in Action*. Arena Verlag, Alderslot, 1983.

[Sch91] K. Schild. A Correspondence Theory for Terminological Logics: Preliminary Report. In *Proceedings of IJCAI-91, 12th International Joint Conference on Artificial Intelligence*, pages 466–471, Sidney, AU, 1991.

[Sch93] D.A. Schön. Learning to Design and Designing to Learn. *Nordisk Arkitekturforskning*, 16(1):55–70, 1993.

[Sch99] U. Schöning. *Theoretische Informatik - kurzgefasst*. Spektrum Akademischer Verlag, 1999.

[SE94] B.N. Sandaker and A.P. Eggen. *Die Konstruktiven Prinzipien der Architektur*. Birkhäuser Verlag, 1994.

[SH95] P. Sachse and W. Hacker. Wie denkt und handelt der Konstruktuer? Jahresforschungsbericht 24, Institut für Allgemeine Psychologie, TU Dresden, Juli 1995.

[SH97] R. J. Scherer and M. Hauser. Perspektiven für die Nutzung der Künstlichen Intelligenz im Bauwesen im Hinblick auf Expertensysteme. In *Perspektiven für die Nutzung der Künstlichen Intelligenz im Bauwesen im Hinblick auf Expertensysteme*, volume Jahrbuch 1997, pages 75–100. VDI-Gesellschaft Bautechnik, Düsseldorf, 1997.

[Shn98] B. Shneiderman. *Designing the user interface: strategies for effective human-computer interaction*. Addison Wesley Longman, 1998.

[Sim77] H. A. Simon. *Models of Discovery*. D. Reidel Pub. Co., 1977.

[Sim95] H.A. Simon. Problem Forming, Problem Finding, and Problem Solving in Design. In A. Collen and W.W. Gasparski, editors, *Design and systems: General applications of methodology*, pages 245–257. Transaction Publishers, New York, 1995.

[Sim96] H. A. Simon. *Models of my Life*. MIT Press, 1996.

[Sim99] H. A. Simon. *The Science of the Artificial*. MIT Press, 1999.

[SML96] C. Schröder, R. Möller, and C. Lutz. A Partial Logical Reconstruction of PLAKON/KONWERK. In *Proceedings of the Workshop on Knowledge Representation and Configuration WRKP-96*, pages 55–64, 1996.

[SMM99] G.M. Sisk, J.C. Miles, and C.J. Moore. A Decision-Support System for the Conceptual Design of Building Structures using a Genetic Algorithm. In A. Borkowski, editor, *Proceedings of 6th EG-SEA-AI Workshop*, Warsaw, Poland, 1999.

[SMM03] G.M. Sisk, J.C. Miles, and C.J. Moore. Designer Centred Development of GA-Based DDS for Conceptual Design of Buildings. *Journal of Computing in Civil Engineering*, 17(3):159–166, 2003.

[SS01] B. Stein and A. Schulz. Modeling Design Knowledge on Structure. In G. Engels, A. Oberweis, and A. Zündorf, editors, *Proceedings of the Workshop of the German Informatics Society, Modellierung 2001*, volume 1, pages 38–48. German Informatics Society, 2001.

[SSS91] M. Schmidt-Schauß and G. Schmolka. Attribute Concept Descriptions with Complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[Ste81] M. Stefik. Planning with Constraints MOLGEN: PART-I. *Artificial Intelligence*, 16(2):111–139, 1981.

[Suc87] L. Suchman. *Plans and Situated Actions: The Problem of Human Machine Communication.* Cambridge University Press, Cambridge, 1987.

[Suh90] N. P. Suh. *The Principles of Design.* Oxford University Press, 1990.

[SW98] D. Sabin and R. Weigel. Product Configuration Frameworks-A Survey. *Journal of the IEEE Intelligent Systems and their Applications*, pages 42–49, jul 1998.

[Thi03] M. Thielscher. The Art and Science of Programming Reasoning Agents. Institute of Artificial Intelligence, University of Dresden, 2003. Lecture Notes.

[TS92] C. Tong and D. Sriram. *Design Representation and Models of Routine Design*. Academic Press London, 1992.

[TW02] H.-P. Tuschik and H. Wolter. *Mathematische Logik - kurz gefasst*. Spektrum Verlag, 2002.

[UT97] Y. Umeda and T. Tomiyama. Functional Reasoning in Design. *IEEE Expert*, 11(4):42–48, 1997.

[UTY90] Y. Umeda, H. Tomiyama, and H. Yoshikawa. Function, Behaviour and Structure. In J.S. Gero, editor, *Applications of Artificial Intelligence in Engineering V: Design*, pages 177–193. Springer Verlag, Berlin, 1990.

[WD01] D. Wilkins and M. DesJardins. A Call for Knowledge-based Planning. *AI Magazine*, 22(1):99–115, 2001.

[Wei96] Robert A. Weida. Closed Terminologies in Description Logics. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, pages 592–599, Menlo Park, August 1996. AAAI Press / MIT Press.

[Wel94] D.S. Weld. An Introduction to Least Commitment Planning. *AI Magazine*, 15(4):27–61, 1994.

[Wel99] D.S. Weld. Recent Advances in AI Planning. *AI Magazine*, 20(2):93–123, 1999.

[Wil88] D.E. Wilkins. *Practical Planning: Extending The Classical AI Planning Paradigm*. Morgan Kaufmann Publishers, 1988.

[Wol03] K. Wolter. Orientierung im Arbeitsprozess der Produktauswahl von komplexen Produkten. Master's thesis, University of Hamburg, Faculty of Computer Science, 2003. Diplom thesis.

[Wom02] O. Wommelsdorff. *Stahlbetonbau-Bemessung und Konstruktion Teil 1*. Werner Verlag, 2002.

[WS92] W.A. Woods and J.G. Schmolze. The KL-ONE family. *Computer and Mathematics with Applications, special issue: Semantic Networks in Artificial Intelligence*, 23(2-5):133–177, March-May 1992.

[WS97] B. Wielenga and G. Schreiber. Configuration-Design Problem Solving. *IEEE Intelligent Systems*, 12(2):49–56, March 1997.

[Wun00] G. Wunsch. *Grundlagen der Prozesstheorie - Struktur und Verhalten dynamischer Systeme in Technik und Naturwissenschaft*. B. G. Teubner Verlag, 2000.

[Yan98] Q. Yang. *Intelligent Planning: A Decomposition and Abstraction Based Approach*. Springer Verlag, 1998.

List of Symbols

A^* Set of action sequences, 47

A_D Set of design actions including test actions and depending on conceptual knowledge K_C, 46

A_P Set of planning actions depending on control knowledge K_P, 46

b Element breadth, 13

B_e Expected benavior of the structural concept, 28

B_s Actual benavior of the structural concept, 28

D Desription of the structural concept in form of sketches, 28

$E_{\mathrm{cm}}I_{\mathrm{c}}$ Flexural rigidity of bracing elements, 19

F Function of the structural concept, 28

f_{cd} Compressive strength of concrete in ULS, 13

F_{Ed} Total vertical force on the building in ULS, 19

g distributed self-weight, 16

h Element height, 18

h_{tot} Total building height, 19

K_C Conceptual knowledge, 46

K_P Control knowledge, 46

l Span length, 13

M_{CD} State model of conceptual design, 46

M_{Ed} Bending moment in ULS, 13

M_{Rd} Bending moment resistance in ULS, 13

n Natural number, 19

N_{Ed} Compression force in ULS, 13

P_{Ed} Concentrated load in ultimate limit state (ULS), 13

q distributed variable load, 16

S Stucture of the structural concept, 28

s system state, 45

T Set of tasks, 46

T^* Set of tasklists, 45

T_C Set of compound tasks, 46

T_S Set of simple tasks, 46

z Internal lever arm, 13

α Lability number, 19

δ Transition function, 45

$\mathcal{G}$ Set of goal system states, 46

$\mathcal{I}$ Intermediate structural concept, 42

$\mathcal{I}_0$ Initial structural concept, 42

$\mathcal{I}_G$ Structural goal concept, 42

$\mathcal{P}$ Action sequence, 47

$\mathcal{S}$ Set of system states, 45

$\mathcal{S}_0$ Initial system state, 45

$\mathcal{S}_G$ Goal system state, 45

$\mathcal{W}$ Design state space, 42

$\mathcal{W}_G$ Set of goal states, 43

$\mu_{\mathrm{Ed,lim}}$ Limiting non-dimensional constant for flexure in ULS, 13

$\vdash$ Design step as transition relation among system states $s, s' \in \mathcal{S}$, 47

$\vdash^*$ Reflexive and transitive closure of a design step, 47