Representation of concurrency in object-oriented design models

Representation of Concurrency in 0bject-0riented Design Models

Convergences dans les modèles de projet orientés-objets

Zur Darstellung von Nebenläufigkeiten in objektorientierten Entwurfsmodellen

Dirk BRETSCHNEIDER
Civil Eng.
Ruhr University
Bochum, Germany

D. Bretschneider graduated in 1992, has worked on information technologies for tunnel rehabilitation. Since 1994, he is involved in research on computer-aided structural steel design. His research concerns realisation of concurrency aspects in object-oriented engineering design systems.

Dietrich HARTMANN
Prof.
Ruhr University
Bochum, Germany

D. Hartmann was Prof. for Struct. Mechanics & Optimisation at the Univ. of Dortmund, 1982-1987. Since 1987, he is Prof. for Computational Eng. at the Univ. of Bochum. His research fields are intelligent computation in CAE, in particular, object-oriented techniques, structural optimisation and parallel computing.

SUMMARY
The complexity of engineering design and construction processes is appropriately managed by applying object-oriented modelling techniques. The consideration of concurrency aspects, inherent to design processes, significantly improves the temporal behaviour and, thus, the acceptance of computer systems based on such techniques. A notation for the explicit representation of concurrency in object-oriented engineering processes is introduced. Subsequently, the power and expressivity of the notation is demonstrated by the description of concurrency phenomena in the process "design and code verification of steel structures according to the German Standard DIN 18800".

RÉSUMÉ
L'étude du projet et de l'exécution en génie civil sont des processus complexes, qui peuvent être maîtrisés à l'aide de techniques basées sur les modèles orientés-objets. La considération systématique des convergences permet d'améliorer considérablement la rapidité et l'acceptation, par l'usager, d'un modèle informatique orienté-objet. La puissance et la facilité d'emploi est illustrée dans le cas de la norme 'projet de calcul et contrôle pour les constructions métalliques selon la norme allemande DIN 18800'.

ZUSAMMENFASSUNG
Die Komplexität ingenieurmässiger Entwurfs- und Konstruktionsprozesse wird sehr gut mit Methoden der objektorientierten Modellierung bewältigt. Durch eine systematische Berücksichtigung inhärenter Nebenläufigkeiten in den modellierten Ingenieurprozessen kann sowohl das Laufzeitverhalten als auch die Akzeptanz eines objektorientierten Computersystems entscheidend verbessert werden. Es wird eine Notation zur expliziten Darstellung von Nebenläufigkeiten in objektorientierten Ingenieuranwendungen vorgestellt. Die Mächtigkeit und Ausdrucksstärke dieser Notation wird anhand der Beschreibung von Nebenläufigkeiten in einer typischen Ingenieuranwendung ("Nachweis und Bemessung von Stahlhochbauten nach DIN 18800") demonstriert.

1. INTRODUCTION

The process from planning and design to construction of structures implies a plentitude of cognitive and procedural activities each associated with a high degree of complexity. For this reason, a consistent integration of these acitivities into an appropriate computer system is very much desirable and benificial. At present, there are various CIM–research projects in civil engineering, funded by the European Union, dealing with integration in computational engineering. By way of example, in the EUREKA project CIMsteel [1] a computer integrated system applicable to the European constructional steelwork industry is developed. In particular, the entire constructional lifecycle from design to manufacturing is taken into account. In the ESPRIT project COMBI [2] emphasis is on the application of advanced information technology methods in order to establish a computer integrated environment for "cooperative" design work. In a further ESPRIT project entitled ATLAS [3] a generic platform for the incorporation of existing "large–scale engineering" application tools is created, also knowledge base representations for a computer integrated design system are examined.

From ongoing research at the ICE[1], it turns out that object–orientation is **the** key technique for future solutions in the CIM–area [4]. The object–oriented paradigm offers an exceptionally suitable platform for the representation of cognitive as well as procedural phenomena within engineering disciplines. Furthermore, it enables programmers to overcome severe difficulties occuring in conventional (procedural) computer science paradigms:

- The object–oriented model of a real–world engineering problem is more natural than the procedural one. This facilitates communication between programmers and application domain experts and provides "better" computer systems.
- The object–oriented approach leads to a much more "reliable" solution than procedural methods because the real–world problem is modeled using visisble and persistent entities of the natural problem domain (the objects with certain characteristics and specific behavior) instead of thinking in abstract and ephemeral procedures.
- Due to concepts like "information hiding" and "encapsulation" the application of object–oriented techniques ensures excellent software reusability compared to conventional solutions and, thus, improves the possibility of future adaptions to state of the art in engineering.

2. BACKGROUND AND PROBLEM DEFINITION

In a present research project, carried out at the ICE and funded by the DFG[2], the individual tasks within structural steel design and construction, such as preliminary design, structural analysis, design and code verification as well as structural detailing, are modeled by applying the object–oriented paradigm. Subsequently, the above four tasks are integrated into a holistic computer system based upon a central object–oriented database management system (OODBMS) in order to support engineers during the entire design as well as construction process. Consequently, the computer system consists of four distinct modules, each one representing one of the four tasks mentioned. All modules are implemented in C++ under the operating system UNIX. The graphical user interface of the computer sytem is based on OSF/Motif, while ONTOS [5] is applied as OODBMS.

In order to manage the complexity of the individual stages in the planning, design and construction process , and to ensure a reasonable response time of the computer system during execution it is obvious that a **concurrent** (parallel or distributed)[3] model is a must. This is a direct consequence of the requirements for integration and holism. Concurrency considerations are a key issue in the development of modern and innovative computer aided engineering systems if the natural way, in which an organized, systematic team work of engineering design experts takes place, is to be modeled: Similar to the human team work a team of computers concurrently performs distributed tasks of the engineering design process. The description and subsequent implementation of concurrency, therefore, is essential for modern and innovative computer aided engineering systems.

In the following an abstract notation for the representation of concurrency phenomena in engineering design processes is introduced. The notation serves (1) to illustrate basic process sequences as well

1. *ICE = Institute for Computational Engineering, Ruhr–University Bochum*
2. *DFG = Deutsche Forschungsgemeinschaft (German Research Foundation)*
3. *The terms "parallelism" and "concurrency" will be used as synonyms throughout the paper.*

as dependencies between single design activities and (2) to detect inherent concurrency aspects – and through this – establishing a framework for a subsequent implementation.

3. CONCURRENCY

Concurrency, as a generic solution concept, is a multi–layered phenomenon. From a programming point of view, four distinct levels of concurrency may be distinguished. Commonly, the following levels of concurrency can be identified: the intra–instruction level (focusing on concurrent operations in individual statements), the inter–instruction level (considering concurrency between program statements), the program–level (concerned with concurrent partial processes within a program) and the job–level (determining concurrent tasks from an overall system view).

A further source of concurrency lies in the various fashions in which instruction streams and data streams consisting of single or multiple items may be processed. Following the classification by Flynn [6] SIMD[4]– and MIMD[5]–hardware architectures may be applied for a computer realization. Recent research in computer technology, however, makes evident that the MIMD–paradigm in association with a message–passing mechanism offers the most powerful and general parallelization platform in engineering disciplines. Therefore, the MIMD–concurrency concept is taken into consideration for concurrent object–oriented modeling within the ICE group, exclusively, and with specific regard to the four parallelization levels in programs mentioned above.

4. OBJECT–ORIENTED MODELING (OOM)

To capture the details of concurrency in engineering applications an abstract notation for its object–oriented representation is an absolute must. An adequate notation has already been introduced by Rumbaugh et al [7]. This notation is based upon an object–oriented analysis (OOA) resulting in a total of three distinct submodels that are interrelated with each other (see Fig.1). Concurrency aspects are particularily represented within the dynamic model that usually follows the establishment of an object model but precedes the definition of the functional model.

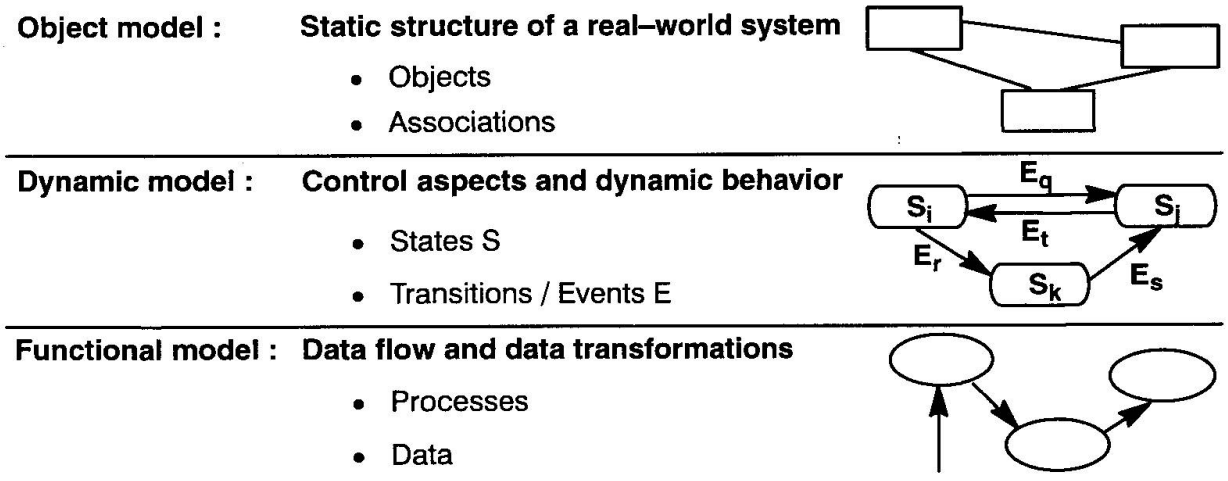

Fig. 1: Submodels of the object–oriented analysis

Since the interactions between the above three submodels are significant for concurrency they are briefly elucidated in the following chapter.

4.1 OOA–submodels

The representation of a real–world problem in terms of objects makes necessary a thorough analysis of the problem domain. To manage the inherent complexity of the real–world problem it is common

4. *SIMD = Single Instruction / Multiple Data*
5. *MIMD = Multiple Instruction / Multiple Data*

practise to decompose the original problem into surveyable portions and to consider only specified aspects of the entire problem in an orthogonal manner.

The object–oriented modeling follows this approach and primarily starts with the objects of the problem domain. This leads to the object model that represents the static structure of the problem by means of its characteristic entities and quantities (objects) as well as the relationships between them (associations). Each object created and defined has specific properties (attributes) and contains specific information on its behavior in terms of operational statements (methods). The object model, mathematically expressed, forms an indirected graph where object classes are represented by means of nodes and relations by edges or lines.

Secondly, the dynamic model is established with a direct reference to the object model. The dynamic model describes the temporal and, therefore, dynamic behavior as well as the control aspects of the problem solution. It is graphically defined through object state diagrams which are directed graphs. The nodes of the graphs are equivalent to states of a specified object while transitions between states are depicted in terms of arrows. Transitions may either be caused by events or happen automatically.

Third, a functional model is created. This model describes the data flow and transformations from object sources to object targets during the execution of the object–oriented computer system. In general, the functional model consists of various data flow diagrams each being a directed graph where nodes correspond with processes and lines (arrows) with data flows.

4.2 Representation of concurrency

As pointed out in the preceding chapter, two fundamental representation concepts are applied within the dynamic model: (object) states and transitions (between states). A state can be understood as an abstract description of a single object with regard to its attribute values and relationships to other objects. A state is valid over a period of time. Commonly, an object, therefore, has multiple states during its lifetime. The states are linked by transitions that have no duration over time. Both, states and transitions of an object are incorporated in the object's state diagram. The entire state diagrams of the different objects form the dynamic model of the system.

For the design of steel structures two characteristic features can be determined:

- most states represent sequences of specified computations , called **operations** in the following,
- transitions are primarily prescribed by the design process, and, therefore, happen **automatically**; only a minority of transitions is caused by explicit events created from direct user interactions.

Consequently, a key point is the detection of concurrency within the above operations. This problem is generally dealt with by Bernstein [8]: Considering two operations each one associated with individual input data and output data, both can only be carried out in parallel when the three conditions shown in Fig. 2 are simultanously satisfied.

$$\text{OUT}(O_1) \cap \text{IN}(O_2) = \{\}$$

$$\wedge\ \text{IN}(O_1) \cap \text{OUT}(O_2) = \{\}$$

$$\wedge\ \text{OUT}(O_1) \cap \text{OUT}(O_2) = \{\}$$

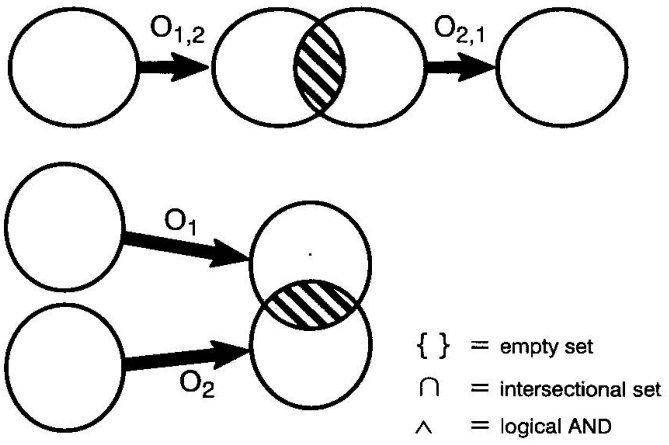

Fig. 2: Bernstein rules

Inherent to the object–oriented modeling are concurrencies due to aggregation. Aggregation is an abstraction principle most frequently used in object–oriented models of engineering disciplines. By

way of example, the decomposition of a structure into single structural components along with a part–whole relationship represents an aggregation (see Fig. 4 in the following chapter). With respect to operations, also aggregate operations can be identified comprising several operations of **one** single object (Fig. 3).

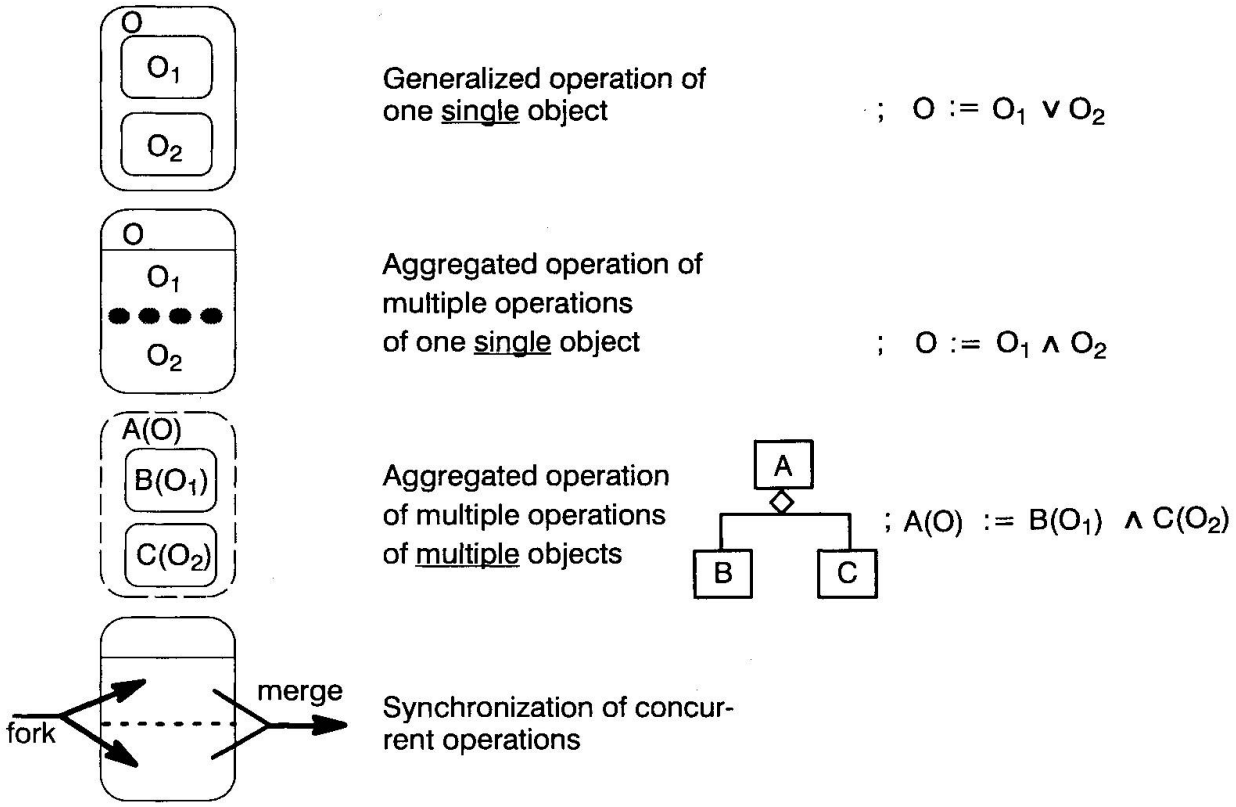

Fig. 3: Advanced concepts of the notation

Furthermore, an aggregate operation may be defined for not one but **many** objects. To prevent confusion in this case, the states have to be precisely identified by a state–name in harmony with the name of the involved object. In addition, the concept of a "generalized state" as well as two synchronization mechanisms for concurrent operations have been defined to provide additional elements for the notation (Fig. 3).

5. DESIGN OF STEEL STRUCTURES ACCORDING TO DIN 18800[6]

5.1 Object model

As mentioned above, the dynamic model encompasses multiple state diagrams, each of which is describing a network of states and transitions between these states for individual object classes. Within the dynamic model, solely classes with significant dynamic behavior patterns are considered. The corresponding classes are grey marked in the object class model of the application considered (see Fig. 4).

In the scope of this paper, exclusively, the dynamic model for "design and code verification of steel structures according to DIN 18800" [9] is examined. Design and code verification according to a given standard is a characteristic activity in the engineering design process and can be a time–consuming process depending on the size of the structure and the number of loading cases. As a consequence, the detection and realization of concurrency significantly improves the temporal behavior and effectiveness of a computer aided "design and code verification assistant" required to cope with future needs.

6. *DIN 18800 = German design code for steel structures [9]*

The dynamic model for "design and code verification" will be clarified through a specific example, where a three dimensional structure is considered that consists of two plane frames connected by two

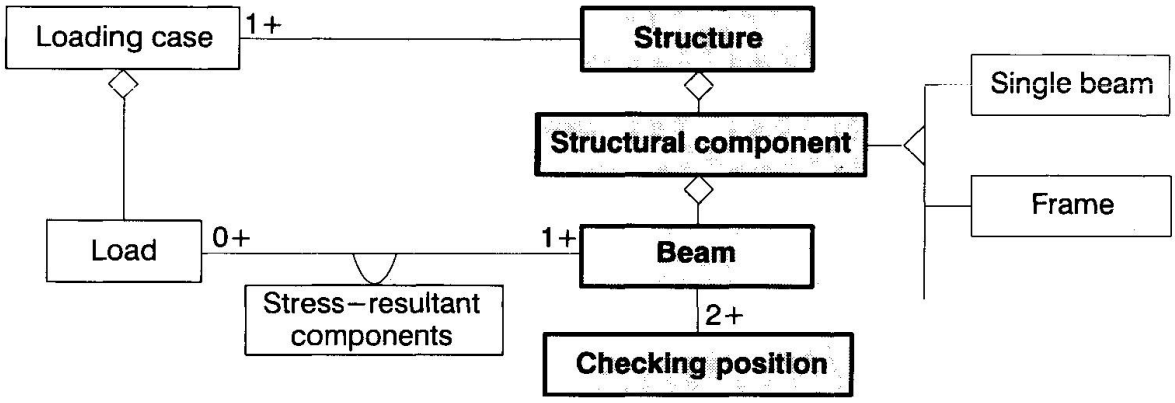

Fig. 4: Partial object model of the engineering application "Design and code verification"

bars perpendicular to the frame planes (Fig 5). The overall structure is represented through an aggregation of four structural parts called "structural components". Each of the structural components contains one or more beams. Usually, a beam has three checking positions where the local feasibility of stresses due to given loads is checked according to DIN 18800[7].

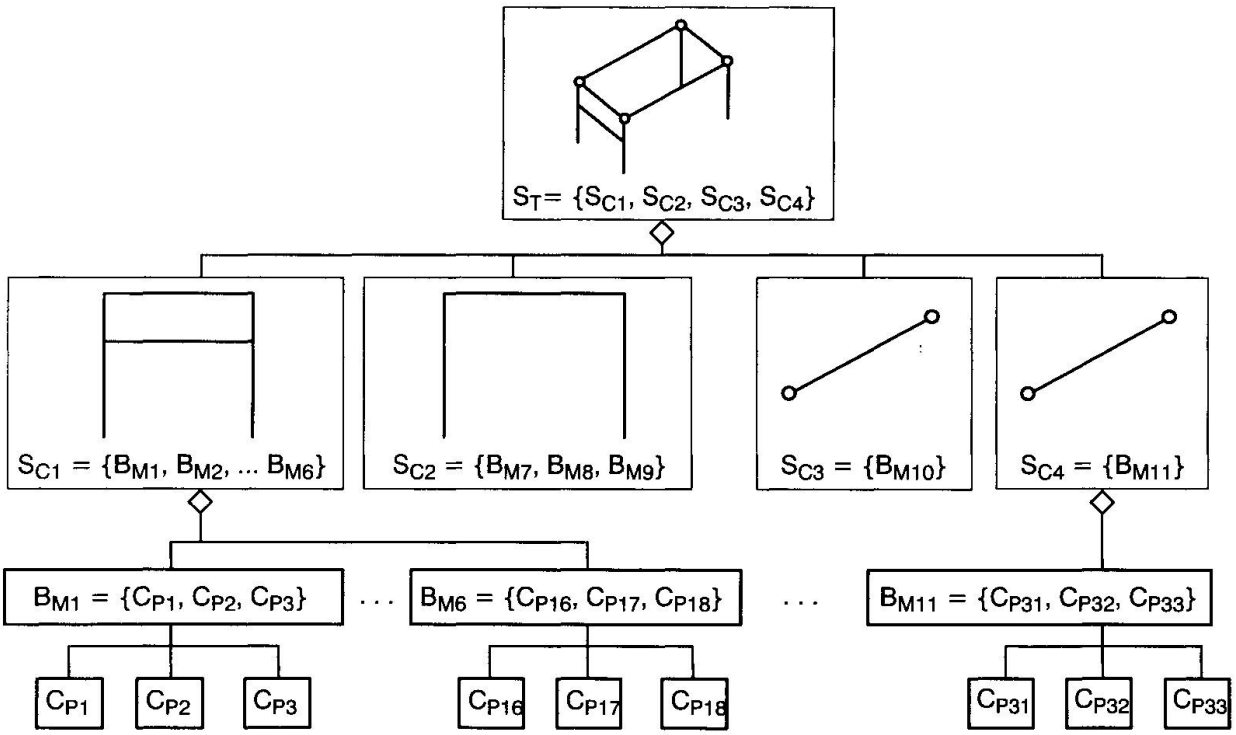

Fig. 5: Partial object instance model for the example considered

The object instance diagram in Fig. 5 used to represent concurrency follows the object class diagram in Fig. 4 when interrelating objects "Structure", "Structural component", "Beam" and "Checking position". In addition to the symbolic notation applied in Fig. 5, relationships between objects are expressed in a mathematical way using sets (e.g. $S_T = \{S_{C1}, S_{C2}, S_{C3}, S_{C4}\}$, indicating that object "Struc-

7. *In general, the number of checking positions in a beam is variable and depends on the number of loading cases as well as the shape and magnitude of different stress–resultant components associated with the loading cases. At present, the design and code verification only encompassses stability phenomena of single structural components but not of total structures.*

ture" (S_T) contains four objects "Structural component" (S_C), called S_{C1}, S_{C2}, S_{C3} and S_{C4}. The relations of the objects "Beam" (B_M) and "Checking position" (C_P) to other objects are represented in a similar way.). The sets serve as a reference to the object model when concurrency phenomena will be discussed in distinct state diagrams of the dynamic model, subsequently.

The object instance model in Fig. 5 has a tree–like structure. Following the tree from top to bottom three levels of concurrency can be identified: (1) concurrency in the structure layer (object S_T), (2) concurrency in the structural components layer (object S_C) and finally (3) concurrency in the checking positions layer (object C_P). All three levels will be examined separately with respect to potential sources of concurrency.

5.2 Representation of dynamic behavior and concurrency

In the overall structure layer the sequence of the three processes "structural analysis", "design and code verification" and "structural detailing" is obvious and demonstrated in Fig. 6. These processes are represented by states "S_T (Structural Analysis)", "S_T(Code verification)" and "S_T(Structural detailing)" of object "Structure", respectivly. All three processes can be integrated in the abstract state "S_T (Structural design)" of the object "Structure" (see Fig. 6).

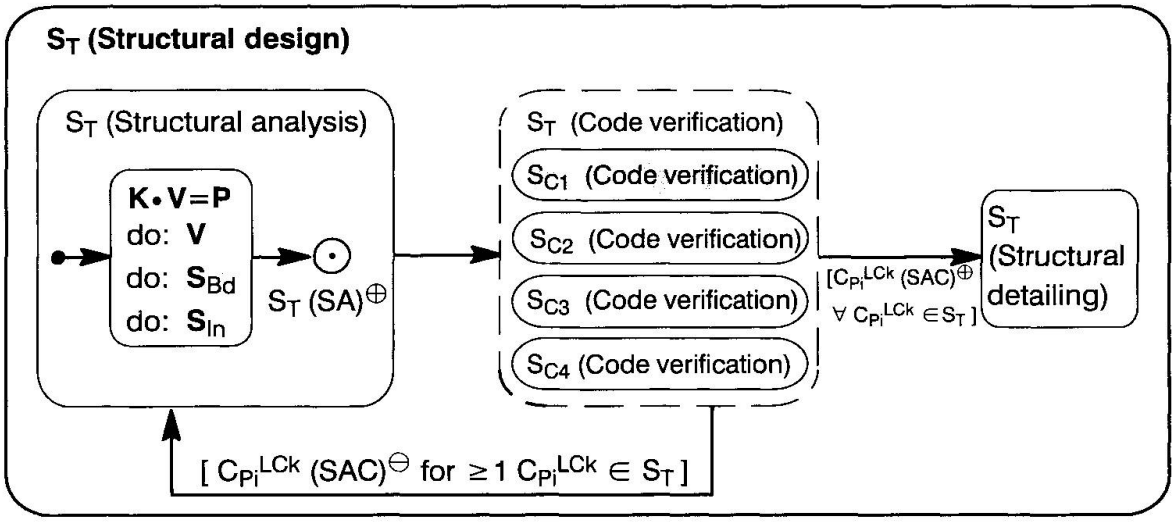

Fig. 6: Concurrency in the structure layer (object S_T)

The state "S_T(Structural analysis)" within state "S_T (Structural design)" is also an abstract state consisting of three substates[8]:

- an initial state that "Structure" automatically represents when entering the structural analysis operation (solid circle),
- a computation state where several activities (indicated by the keyword "do:") are performed, particularily the solution of the stiffness equations K•V=P is carried out and leads to the displacements V and the stress–resultant components S_{Bd} and S_{In} of the structure,
- subsequent to the computation state "Structure" automatically enters its final state (bull's eye), called "$S_T(SA)^{\oplus}$"; the plus sign indicates that the structural analysis (SA) of the entire structure S_T has been performed successfully.

Then the state "S_T(Code verification)" is entered. This state is an aggregate operation of several suboperations, each performed by a distinct object: The notation in Fig. 6 specifies that "design and code verification" of the whole structure S_T can be performed in parallel for every agggregated structural component $S_{Ci} \in S_T$. Hence, the state "S_T(Code verification)" is a "nested state" where substates may be expanded in separate state diagrams.

8. *Only those aspects within state "S_T(Structural analysis)" are pointed out that express dependencies to the state of main interest, "S_T(Code verification)". A more detailed consideration of "S_T(Structural analysis)" and "S_T(Structural detailing)" is omitted, because emphasis is laid on "S_T(Code verification)" within this paper.*

The state "S_T(Code verification)" is finished when the stress analysis has been performed at every checking position C_{Pi} for every load case LC_k. According to the verification results state "S_T(Structural analysis)" or state "S_T(Structural detailing)" is entered. This is graphically shown through arrows in Fig. 6 representing "guarded transitions". By that object "Structure" is transfered from state "S_T(Code verification)" to another state if the individual guard condition shown as a Boolean expression in brakkets is true. Thus, if the stress analysis is verified at all checking positions of the structure (indicated by condition "$[C_{Pi}{}^{LCk}(SAC) \oplus \forall\, C_{Pi}{}^{LCk} \in S_T]$" in Fig. 6) the structure can be designed in detail. Consequently, the state "S_T(Structural detailing)" is entered. By contrast, in the case that the stress analysis is not verified at one or more checking positions (indicated by condition "$[C_{Pi}{}^{LCk}(SAC)^{\ominus}$ for $\geq 1\ C_{Pi}{}^{LCk} \in S_T]$" in Fig. 6), the corresponding parts of the structure have to be changed. This, of course, makes a re-analyzation of some parts or the entire structure necessary. Thus, state "S_T(Structural analysis)" is re-entered again (see Fig. 6).

As mentioned previously besides concurrency in the structure layer there also exists concurrency in the structural components layer. This is exemplified by expanding the state "S_{C1}(Code verification)" that is grey-shaded in Fig. 6. Examining a single structural component, basically three groups of operations can be parallelized when performing the stress analysis:

- computation of the effective length of all associated beams by determining the buckling factors β (state "Buckling factor (β)"),
- computation of the allowable stress-resultant components representing yielding in the individual cross section S_{pl} (state "Allowable plastic stress-resultant components (S_{pl})"),[9]
- execution of the stress analysis to verify that the collapse load is not exceeded at every checking position C_{Pi} and for every load case LC_k given (state "Stress Analysis against Collapse (SAC)").

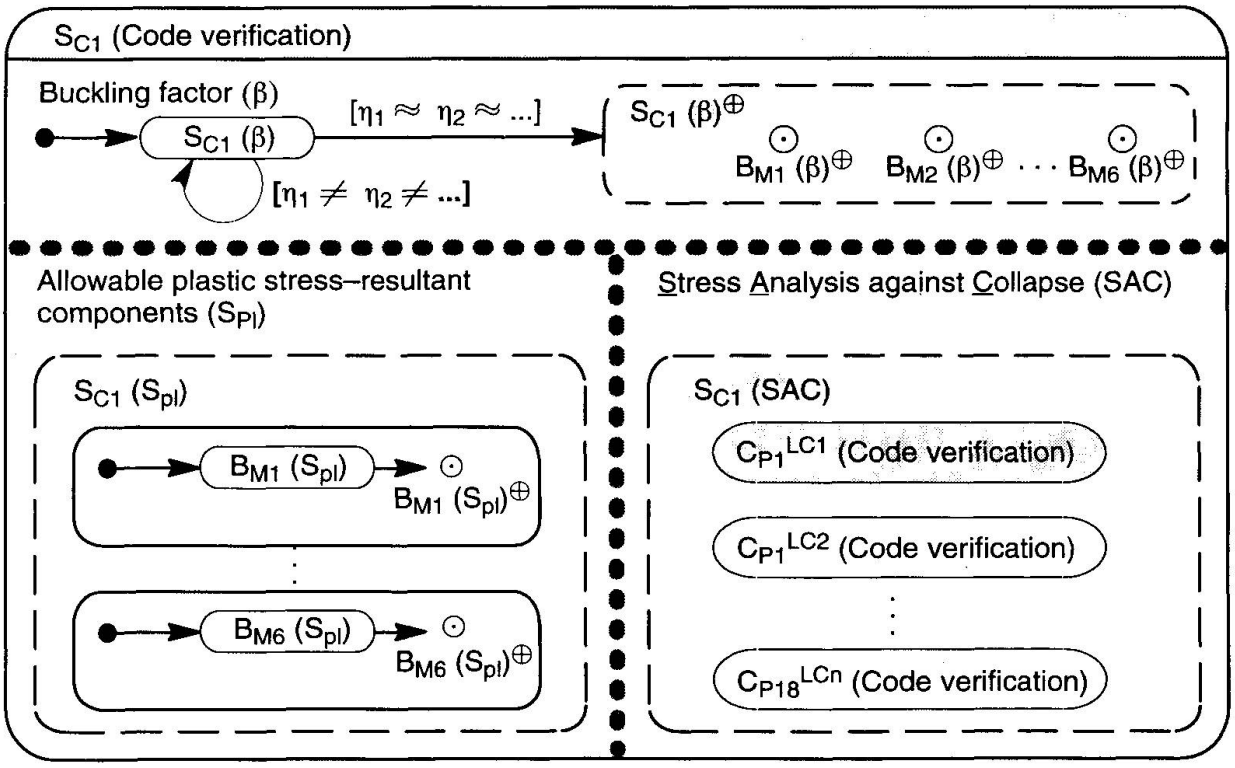

Fig. 7: Concurrency in the structural component layer (e.g. object S_{C1})

The concurrency incorporated between these three states is clarified by dotted lines inside state "S_{C1} (Code verification)" (see Fig. 7).

9. *The german code DIN18800 offers three different possibilities to compare loading and loading capacity, thus, to verify a structure: elastic-elastic, elastic-plastic, plastic-plastic. The considerations in this contribution are based on the method elastic-plastic: Loading, expressed in stress-resultant components, is based on elastic computations. Loading capacity, expressed in stress-resultant components for plastification in the cross section, is based on plastic computations.*

In addition, the three individual states themselves are again superstates entailing concurrent substates. These are either substates, describing operations of the **same** object (e.g. the computation of the buckling factor "$S_{C1}(\beta)$") or operations of **aggregated** objects expressed by rounded boxes with dashed lines (e.g. the computation of allowable yield stresses "$B_{Mi}(S_{pl})$" expressed in stress–resultant components for every beam associated, or the execution of the stress analysis for the specified checking position and load case "$C_{Pi}{}^{LCk}$ (Code verification)").

In some cases it is important to ensure that computations inside states are terminated, because the computation results are required in subsequent states. Thus, final states have to be defined, e.g. the state $B_{M1}\ (S_{pl})^{\oplus}$ indicating that all allowable yield stresses of beam 1 expressed in stress–resultant components have been computed (Fig. 7).

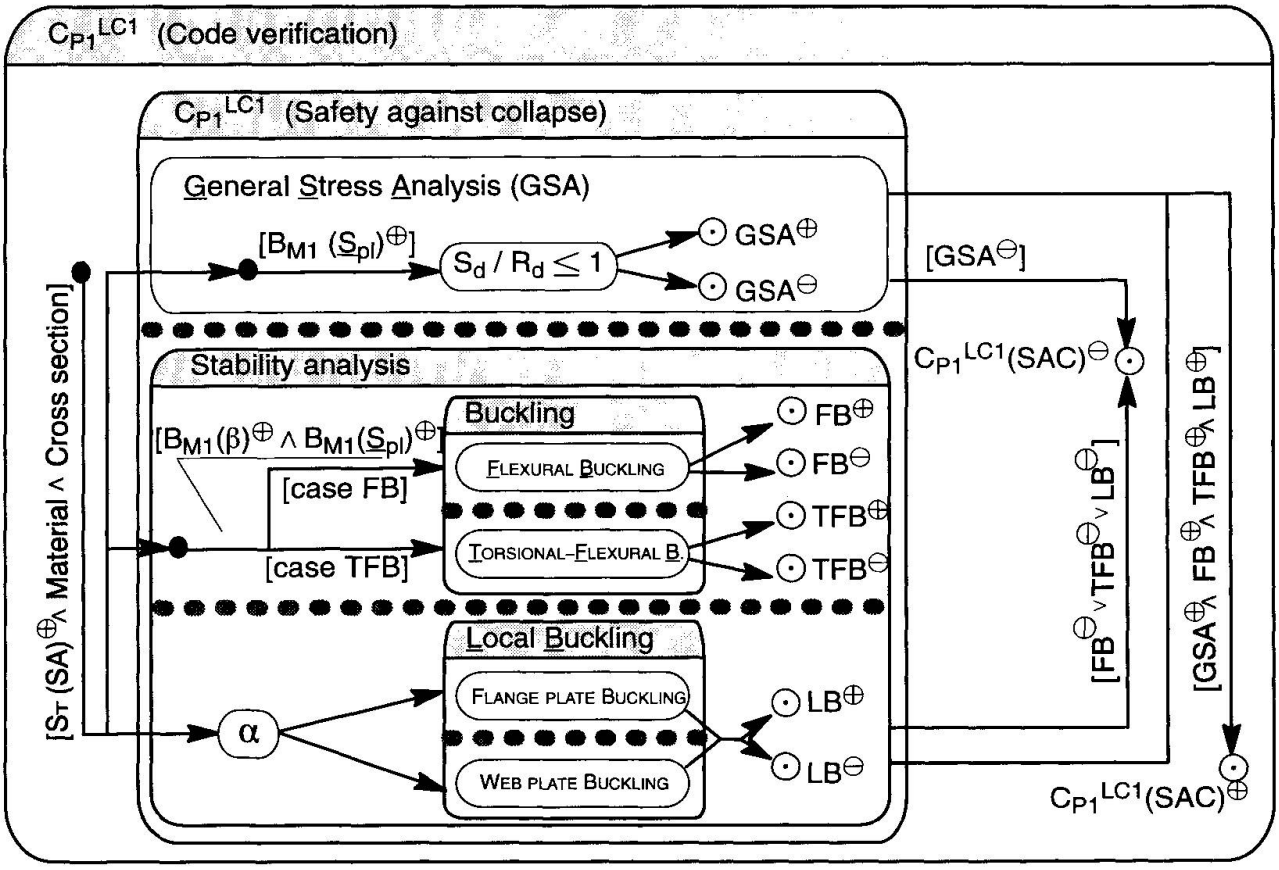

Fig. 8: Concurrency in the checking positions layer (e.g. object C_{P1})

Concurrency in the checking positions layers is examined by way of an example, too. Considering the state "$C_{P1}{}^{LC1}$ (Code verification)" this state is expanded as illustrated in Fig. 8. In this case, the stress analysis at specified checking positions can be described by means of three substates:

- an initial state that is finished automatically if structural analysis results ($S_T\ (SA)^{\oplus}$), material's and cross section's properties are known,
- a computation state through which safety against collapse with regard to different aspects (e.g. flexural buckling, torsional flexural buckling etc.) is determined (state "$C_{P1}{}^{LC1}$ (Safety against collapse)"),
- two alternative final states "$C_{P1}{}^{LC1}(SAC)^{\oplus}$" and "$C_{P1}{}^{LC1}(SAC)^{\ominus}$" entered automatically depending on the results of the code verification; the final state affects the process sequence within state "S_T(Structural design)" as shown in Fig. 6.

Inside state "$C_{P1}{}^{LC1}$ (Safety against collapse)" several code verification procedures for various categories of collapse are carried out. In detail these are a general stress analysis (GSA), verification of

safety against flexural buckling (FB) and torsional–flexural buckling (TFB) as well as verification of stability against local buckling, separately for beam's web and flange plate (LB). All five code verifications can be examined in parallel. The concurrent portions are elucidated by dotted lines (see Fig. 8).

The transitions between the initial state and the stress analysis as well as the final states require measures for synchronization as introduced in chapter 4.2. State "$C_{P1}{}^{LC1}$ (Safety against collapse)" basically has three independent substates. If the guard condition at the guarded transition from the initial state to the stress analysis state "$C_{P1}{}^{LC1}$ (Safety against collapse)" is satisfied all three substates are active at the same time. This requires a fork of the control flow. Contrary to this, the transition from the stress analysis state to the final state "$C_{P1}{}^{LC1}$ $(SAC)^{\oplus}$" needs a merge of the control flow because the code verification at checking position C_{P1} for loading case LC1 can only be carried out if the constraints of **all** distinct code verification procedures are fulfilled. On the other hand, the transition from the stress analysis state to the final state "$C_{P1}{}^{LC1}$ $(SAC)^{\ominus}$" needs no synchronization because the violation of one single constraint leads to state "$C_{P1}{}^{LC1}$ $(SAC)^{\ominus}$".

6. CONCLUSIONS

The abstract notation introduced in this paper provides adequate expressive power along with a high representation density in describing concurrency of object–oriented models in engineering disciplines. Conceptionally, also parallelisms in the inter–instruction level between single statements can be captured using the concept of nested states.

In this contribution, particularily, all candidate categories of concurrency in the specific domain of "design and code verification of steel structures according to DIN 18800" are demonstrated. In a prototype computer system created at present, however, only portions of the overall concurrency concept are materialized in a coarse–grain solution. The prototype primarily models the natural concurrency of the team work of engineering design experts with sufficient success (see chapter 2). With respect to the concurrent implementation basically two categories of hardware have been applied at the ICE: (1) a highly parallel machine (transputer system) and (2) a heterogenous network of UNIX–workstations connected by a standard Ethernet. Subsequent to an evaluation of both hardware platforms the coarse–grain implementation model is realized on the workstation cluster applying PVM [10] as a message passing interface and using C++ as implementation language.

ACKNOWLEDGEMENTS

The authors would like to thank the DFG, Bonn, for funding this research work since 1992. The support by the DFG enabled the ICE research group to scrutinize a new and sophisticated field of research within object–oriented modeling.

REFERENCES

[1] CIMSTEEL, Computer Integrated Manufacturing for Constructional Steelwork, Information Brochure on the EUREKA 130 research project, Taylor Woodrow Construction Holding Ltd., 1994

[2] SCHERER et al., Architecture of an Object–Oriented Product Model Prototype for Integrated Building Design, Contribution to the 5th Int. Conference on Computing in Civil and Building Engineering, Anaheim, CA, USA, June 1993

[3] GARAS et al., Presentation of the EU ESPRIT research project "ATLAS", Session EP2 of the 1st European Conference on Product and Process Modeling in the Building Industry, Dresden, DE, Oct. 1994

[4] HARTMANN et al., Object–Oriented Modeling of Structural Systems, Contribution to the 5th Int. Conference on Computing in Civil and Building Engineering, Anaheim, CA, USA, June 1993

[5] ONTOS DB 2.2 Reference Manual, Vol. 1 "Class and Function Libraries", ONTOS Inc., 1992

[6] FLYNN, Some Computer Organizations and their Effectiveness, IEEE Transactions, Vol. C–22, 1972, pp. 948–960

[7] RUMBAUGH et al., Object–Oriented Modeling and Design, Prentice–Hall Int. Inc., 1991

[8] BERNSTEIN, Program Analysis for Parallel Processing, IEEE Transactions, Vol. EC–15, Oct. 1966

[9] DIN 18800 "Stahlbauten", Teile 1 und 2, Ausgabe November 1990

[10] GEIST et al., PVM 3 – Parallel Virtual Machine User's Guide and Reference Manual, Technical Report ORNL/TM–12187, Oak Ridge National Laboratory, USA, 1993