

**REPRESENTATION OF A REINFORCED CONCRETE DESIGN  
CODE AS AN OBJECT ORIENTED MODEL**

**M.Phil. Thesis**

**W J B Shiromal Fernando**

**UNIVERSITY OF MORATUWA  
SRI LANKA**

**May 2010**

**REPRESENTATION OF A REINFORCED CONCRETE DESIGN  
CODE AS AN OBJECT ORIENTED MODEL**

**by**

**W J B Shiromal Fernando**

**A thesis submitted to University of Moratuwa**

**for the Degree of Master of Philosophy**



**Research supervised**

**by**

**Professor Priyan Dias**

**DEPARTMENT OF CIVIL ENGINEERING**

**UNIVERSITY OF MORATUWA**

**MORATUWA**

**SRI LANKA**

**May 2010**

## ABSTRACT

Design standards comprise many knowledge types such as text, rules, equations, tables, graphs and figures. The attempt is to encode the standard without distorting the format of the standard, i.e. to represent the standard clauses and tables in the same format as in the standard. This effort will facilitate changes to the standards without much variation to the programme code.

This thesis presents a framework to model standards using the Object Oriented Programming paradigm. It also presents the concept of a common interface, i.e. to accommodate several design standards for reinforced concrete design in one module; however, implementation is carried out only for BS8110. The programme uses an inferencing mechanism for execution, which is a similar method of execution to that of a standard's user; it is not a hard coded structured programme. This is a novel concept when compared to the available software for reinforced concrete design.

The literature review investigates the structure of typical standards and the available standards processing technique such as Predicate Logic, Decision Tables, Production Systems and Semantic Networks before choosing Object Oriented programming as the preferred one. The review also compares both the provisions and design outputs of several reinforced concrete standards.

The Common Interface for Design Standards (COIDS) has three main modules (or models), namely the Product Model, Standards Model and Interaction Model. The Product Model handles the product data, e.g. Frame Data. The Standards Model handles the standards data, i.e. it contains all the knowledge in a standard. The Interaction Model handles the data exchange between the user, COIDS objects and external software. It transfers data from the COIDS to external analysis software and maps analysis output files to COIDS. An Object Oriented Shell called KAPPA was used to develop the object oriented model.

## ACKNOWLEDGEMENT

I am most grateful to my research supervisor, Prof. Priyan Dias, Head of Civil Engineering, University of Moratuwa for selecting me for carryout this research on Representation of a Reinforced Concrete Design code as an Object Oriented Expert system. He guided the research work presented in this thesis with much dedication and enthusiasm. I also wish to thank him for his valuable advice and the tedious task of correcting the study, helping me on weekends sacrificing holidays. If not for his continuous persuasion and encouragement, I would not have completed this thesis with my office work load.

I wish to thank the members of the progress review committees, Prof. Saman Bandara, Dr. (Mrs.) Premini Hettiarachchi, Prof. SAS Kulathileke, Prof. UGA Puswewala and Dr. Ruwan Weerasekara for their valuable advice.

I would also like to thank Nuwan Kodagoda for the initial work he had carried out on this research. I have incorporated some of his initial work in this thesis.

I would also like to thank Prof. Lakshman Alwis, Chairman, Design Consortium Limited, for granting duty leave for my research work. I would also thank my DCL colleagues for continuously supporting me with my office work, during the final stages of the research.

Finally, I would like to thank my family members for the support and encouragement, specially our parents for looking after my domestic responsibilities, without their support I would not have been able to finish this work.

## **DECLARATION**

This thesis is a report of research carried out in the Department of Civil Engineering, University of Moratuwa, between July 1996 and December 2009. Except where references are made to other work, the work has not been submitted in part or whole to any other university. This thesis contains 102 pages.

Welisera Jude Basil Shiromal Fernando

Department of Civil Engineering

University of Moratuwa

## LIST OF FIGURES

Fig.2.1 Prolog Implementation of a Table .....	16
Fig.2.2 Prolog Implementation of Calculation of Tension Steel area in a Beam .....	17
Fig.2.3 Calculation of $(x/d)_{lim}$ using a decision table.....	17
Fig.2.4 Calculation of Longitudinal Spacing for shear using a decision table .....	18
Fig.2.5 Production System Implementation of Calculation of Tensile Steel area in a Beam .....	19
Fig.2.6 Details of Backtracking through a rules given in fig. 2.5.....	20
Fig.2.7 Semantic network which shows the relationship of physical objects in a design standard .....	21
Fig.2.8 Semantic network that shows details of one particular object.....	21
Fig.2.9 Knowledge Representation in a hierarchy of frames that show inheritances..	22
Fig.2.10 Frame of a Beam.....	23
Fig.2.11 Data Abstraction Levels of Building Structures.....	25
Fig.2.12 Object oriented representation of Parent Class and the Child Class .....	26
Fig.2.13 Object oriented representation of single inheritance and the multiple inheritances .....	27
Fig.2.14 Object oriented representation of object instances .....	27
Fig.2.15 Object Orientated representation of Objects, which includes both slot value (attribute) and the method to evaluate the attribute (Encapsulation).....	28
Fig.2.16 Object oriented representation of concept polymorphism, A single message to carryout multiple task .....	29
Fig.3.1 Part of the Contents (Section 3) of BS8110 Part 1 (1997) structured based on element type .....	31
Fig.3.2 Part of the Contents (Section 6) of Euro Code 2 (BS EN 1992-1-1:2004) structured based on stress states .....	31
Fig.3.3 Definition of Dimensions (Figure 2.2 of EC2).....	35
Fig.3.4 Approximate effective span for calculation of effective breadth ratio (Figure 2.3 of EC2) .....	35
Fig.3.5 Stress block used in EC2 is compared with that in BS110.....	38

Fig.3.6 Service Loading arrangement (Example 7, of Graded Example in reinforced concrete design by W.P.S.Dias, 1998).....	40
Fig.4.1 Conceptual Model of the COIDS .....	43
Fig.4.2 KAPPA representation of the Conceptual Model of the COIDS as an Object Hierarchy.....	44
Fig.4.3 Product Data Object Hierarchy.....	45
Fig.4.4 Typical Element Instance .....	46
Fig.4.5 Standards Data Object Hierarchy .....	47
Fig.4.6 Basic Data Object Hierarchy .....	48
Fig.4.7 Sub Class Durability_BS .....	48
Fig.4.8 Common Steel Class.....	49
Fig.4.9 Steel_BS Sub Class .....	50
Fig.4.10 Lotus 123 representation of the BS Table 3.4 (BS34).....	51
Fig.4.11 Class Table_BS and the Method BS34 look up routines.....	51
Fig.4.12 Derived Data Object Hierarchy .....	52
Fig.4.13 Class Clause Object which includes the common slots clauses .....	53
Fig.4.14 Sub Class Clause_BS Object, slots for inferencing and BS 8110 clauses as methods .....	53
Fig.4.15 KAPPA Method BS3415 represents the BS8110 clause 3.4.1.5 .....	54
Fig.4.16 CLAUSE_BS effective_span slots .....	54
Fig.4.17 Sub Class Symbol BS, which handles the Symbols data Items in the Object's Slots and the corresponding methods in the Object's Methods .....	55
Fig.4.18 Interaction Model Object hierarch.....	56
Fig.4.19 KAPPA Function's Input Frame Data Method .....	57
Fig.4.20 Class User Query Methods .....	58
Fig.4.21 Durability Data dialog box posted to the user by the User Query object.....	58
Fig.4.22 Class Mapping .....	59
Fig.4.23 COIDS Graphical User Interface.....	60
Fig.4.24 Class Draw Data .....	60
Fig.4.25 Class Data Item Network.....	61
Fig.4.26 Class Data Item Network, slot Check Condition store flags to identify the check state.....	62
Fig.4.27 Class Processed Data .....	63

Fig.4.28 Class Beams.....	64
Fig.4.29 Class Beams_BS.....	64
Fig.5.1 KAPPA Main Window with seven Icons.....	67
Fig.5.2 KAPPA’s Main Object hierarchy.....	68
Fig.5.3 KAPPA’s Extended Object Hierarchy which indicates the image object hierarchy.....	69
Fig.5.4 COIDS Object Connected to the Class Root.....	70
Fig.5.5 KAPPA Active Images Tool Box.....	71
Fig.5.6 KAPPA Active Images package.....	71
Fig.5.7 KAPPA Knowledge Tool Window.....	72
Fig.5.8 KAPPA Rules linked to the COIDS standards clause BS3444.....	74
Fig.5.9 COIDS strategy to execute standards clauses using KAPPA rules.....	74
Fig.5.10 COIDS Goal “CheckCondition” to find the stress state of the element to be checked.....	75
Fig.5.11 Instruction flow from KAPPA Functions to COIDS (Stage 1).....	76
Fig.5.12 Data Item Network Object sends a message to forward chain the KAPPA rules (Stage 2).....	77
Fig.5.13 Forward Chaining KAPPA Rule Beams will execute the method Get_ CheckBeamData to get data from user (Stage 3).....	78
Fig.5.14 Forward Chaining rule CheckBSBeams will calculate the requirement specified by the standard (Stage 4).....	78
Fig.6.1 Common Interface Session Window.....	79
Fig.6.2 Basic Data Input Session Window.....	80
Fig.6.3 KAPPA Product Model Hierarchy.....	81
Fig.6.4 KAPPA Function Editor INIT method will send messages to COIDS Objects.....	81
Fig.6.5 Basic Data Input Session Window, Initialise button will initialise the COIDS.....	82
Fig.6.6 Basic Product Data Dialog Box.....	83
Fig.6.7 Product Model hierarchy with the Instances generated based on the frame data user input.....	83
Fig.6.8 Typical Node Data User Input.....	84
Fig.6.9 Display User Node Inputs.....	84
Fig.6.10 Typical Element Data User Input.....	85

Fig.6.11 Display User Element Inputs .....	85
Fig.6.12 Typical Support Data User Input.....	86
Fig.6.13 Typical Section Property Data User Input.....	86
Fig.6.14 Element number to input Element Load.....	86
Fig.6.15 Number of Load types on the Element.....	86
Fig.6.16 Load types on the Element to be defined.....	87
Fig.6.17 Typical Uniform Load Input data.....	87
Fig.6.18 Typical Material Data User Input.....	88
Fig.6.19 Typical Material Data User Input for concrete.....	88
Fig.6.20 Typical Durability Data User Input for Fire Resistance.....	89
Fig.6.21 Typical Durability Data User Input for Exposure Condition .....	89
Fig.6.22 User Define Path of the Analysis Package executive path.....	90
Fig.6.23 MICROFEAP-11 Analysis Software.....	90
Fig.6.24 User to insert the element number.....	91
Fig.6.25 User to insert which standard to be used for checking the element.....	92
Fig.6.26 Ele_9 Instance is generated based on the user input.....	92
Fig.6.27 User to input the stress states or Serviceability Limit States to be checked..	92
Fig.6.28 User to input the path of the Analysis Data File.....	93
Fig.6.29 Message to the user by COIDS Indicating the Mapping is complete.....	93
Fig.6.30 User to input the percentage redistribution.....	93
Fig.6.31 Reinforcement data at section 1 (Support) .....	94
Fig.6.32 Reinforcement data at section 2 (Middle Section) .....	94
Fig.6.33 Reinforcement data at section 3 (Support) .....	95
Fig.6.34 Ele_9 Instance which included the processed data-items.....	95
Fig.6.35 COIDS message to user Section 1 is satisfactory in Flexure .....	95
Fig.6.36 COIDS message to user Section 1 satisfactory in Shear .....	96
Fig.6.37 COIDS message to user regarding shear reinforcement.....	96
Fig.6.38 COIDS message to the user stating that the Task is completed .....	96

## **LIST OF TABLES**

Table 3.1 Structure and Philosophy of Design Standards.....	32
Table 3.2 Recommended Live loads.....	33
Table 3.3 Recommended basic load safety factors.....	33
Table 3.4 Recommended Material Properties.....	34
Table 3.5 Recommended Material safety factors.....	34
Table 3.6 Recommended equations to calculate the flange width of a beam .....	35
Table 3.7 Recommended pattern loading in BS8110 and EC2 .....	37
Table 3.8 Recommended stress block diagrams by other standards.....	39
Table 3.9 Summary of output of the design example .....	41

## **LIST OF KEY WORDS**

Common Interface, Expert Systems, Knowledge Base, Object Oriented Programming, Standards Processing, Reinforced Concrete Design

## TABLE OF CONTENTS

Abstract.....	i
Acknowledgements .....	ii
Declaration .....	iii
Table of Contents .....	iv
CHAPTER 1 – INTRODUCTION .....	1
1.1 Significance of Research .....	1
1.2 Objectives .....	4
1.3 Methodology .....	5
CHAPTER 2 – DESIGN CODES AS EXPERT SYSTEMS .....	7
2.1 Introduction .....	7
2.2 Design Standards .....	7
2.3 Aim of Building Standards.....	8
2.4 Properties of Standards .....	8
2.5 Expert Systems.....	9
2.6 Model-Based Reasoning .....	10
2.7 Design Standards as Expert Systems .....	12
2.8 Standards processing techniques .....	14
CHAPTER 3 – COMPARISON OF DESIGN STANDARDS.....	30
3.1 Introduction .....	30
3.2 The Structure of Design Standards.....	30
3.3 Basis of Design .....	32
3.4 Loads, Load Combinations and Partial Safety Factors.....	32
3.5 Material Properties .....	33
3.6 Physical Geometry of Structures .....	35
3.7 Other Key aspects of design standards .....	36
3.8 Design Example .....	40

CHAPTER 4 - COMMON INTERFACE CONCEPT AND IMPLEMENTATION ..	42
4.1 Introduction .....	42
4.2 Product Model.....	44
4.3 Standards Model .....	46
4.4 Interaction Model.....	56
CHAPTER 5 – KAPPA APPLICATION DEVELOPMENT SOFTWARE .....	65
5.1 Introduction .....	65
5.2 The KAPPA Interface .....	67
5.3 Knowledge Processing Techniques in KAPPA .....	72
5.4 Execution process of COIDS in Checking Mode.....	75
CHAPTER 6 – TYPICAL COIDS SESSIONS .....	79
6.1 Data Input Mode .....	79
6.2 Analysis Mode .....	89
6.3 Checking Mode.....	90
CHAPTER 7 – CONCLUSIONS AND RECOMMENDATIONS.....	97
REFERENCES .....	99

# CHAPTER 1

## INTRODUCTION

### 1.1 Significance of Research

Buildings are designed to commonly accepted norms by the engineering community to ensure safety and sustainability. Design standards adopted by different engineering communities differ based on their experience and practice. The standards represent expert knowledge compiled over a long period of time. The standards are updated regularly with new knowledge. The knowledge is mostly stipulated by mandatory rules. This knowledge representation makes standards to be an ideal knowledge base to be utilized to build an expert system.

Standards are often voluminous. British standard BS8110: Part1 (1997) “Structural Use of Concrete in Buildings” is 164 pages long. There are additional parts to BS8110 each of similar size. BS5950 (2000): “Structural use of Steel” consists of nine parts, each a separate document. In addition there are cross references to the other British Standards such as loading for buildings (BS6399), which in turn has three more parts. This volume of information involved requires the investment of considerable time and effort in becoming accustomed with a particular code (Neilson, 1997). This effort will not be productive if an engineer needs to use another standard for his next assignment, since the knowledge is not retained.

Let’s look at the history of the development of Eurocodes to understand the process of development, the effort and the time period required to evolve a new standard. According Menzies and Gulvanessian (1998), a proposal to develop an international set of codes of practice for structural design was first agreed in 1974 by several technical-scientific organisations based largely in Europe. Following preparatory work by these organizations, the Commission of the European Communities (CEC) together with the European Free Trade Association (EFTA) took the initiative for developing the Structural Eurocodes at the end of the 1970s by establishing a steering committee to oversee the work. In 1989 the responsibility for their development was transferred to the European Committee for Standardisation (CEN). Since English was the most widely spoken and understood language, a decision was made in 1991 to conduct

meetings and to prepare the Eurocodes in English in the first instance. Once each Eurocode Part is approved for publication as a pre-standard (ENV) it is translated into the other two official CEN languages, French and German, and published in the three languages. A programme to convert the ENV Eurocodes to European standards (EN) has now been completed.

The issue of European standards will lead to the withdrawal from use of the national codes of practice in the different European countries which are members of CEN. There is a period of five years during which national codes are to co-exist with the European standards “Eurocodes” before the national codes are to be withdrawn by the year 2010.

Engineers will take at least another two to three years to become familiarized with the new standards. By that time there will be new amendments proposed by the standards committees with the new social requirements, international commitments and use of new materials for buildings. A possible amendment will be based on environmental issues and emergence of new nano materials.

For a new standard to evolve, it will take at least twenty to thirty years. Upgrading existing standards will be carried out every two to five years. The effort is wasted if the standards are not utilized to the maximum. This effort should encourage the engineers to utilize the standards knowledge to the maximum and this will also result in safe and economical designs.

The engineer needs to refer many design standards for his day to day work. The knowledge in design standards is under-utilized due to the complexity of the knowledge representation of standards and the knowledge gap between the standard’s knowledge and that of the average engineer. This aspect will also result in a longer adaptation period to implement a standard with new knowledge or using a less familiar standard (For example, an engineer familiar with British standards using an American standard).

As a practicing structural engineer, the author’s estimate is that less than ten percent of a standard’s knowledge is utilized by engineers. Junior engineers will follow

standard clauses and equations without much understanding of the principles in the code. Such misinterpretation of code clauses will lead to unsafe designs. Senior engineers on the other hand hardly refer the code since their experience will assist them to make judgements for routine designs. In new situations, time will not permit a senior engineer to go through all the standard clauses and their references. Thus they will tend to over design the sections; this will lead to uneconomical design. Thus we see that the knowledge in a standard is under-utilized by both junior and senior engineers, leading to potentially unsafe or uneconomical designs respectively. Computerized modelling of design standards will contribute towards remedying this situation. If the model is developed to the level of an expert system that can be queried for explanations, then the system will serve not merely to fill the gap in engineers' knowledge of standards, but to educate them as well.

#### **1.1.1 Drawbacks associated with available design software**

Computer software is used by engineers for analysis and design work. Most engineers use one software to analyse and another software for design. There are software packages that claim to perform both analysis and design, complying with many international standards. However there are the following drawbacks associated with such software packages;

- The design software available is mostly a collection of templates that are defined to carry out a particular task such as beam, column or foundation section design. They are sometimes developed using electronic work sheets.
- The above mentioned software is developed using hard coded structured programming techniques. Thus incorporation of revisions to the standards can be done only by major revision to the programming code.
- Since the programmes are hard coded, there are many assumptions being made by the programmer, such as effective span of a beam being centre to centre distance of the supports, beam not being subject to axial force or torsion etc.. In most of the programmes, the engineer needs to input code data to the programme, such as cover based on the environmental conditions and fire requirements. Thus the programme plays the role of a mere calculator with hard coded equations.

- Although some programmes claim to contain both analysis and design facilities to many international standards, practically speaking we use them only for the conceptual design stage but not for the detailed design of elements, since they do not carry out all the design checks specified in standards, specially the serviceability checks. These programmes also have attached hard coded templates to the analysis programme which carry out pre-defined procedures. Due to these reasons most designers are compelled to use separate programmes for analysis and designs.

The approach presented in this work is a more fundamental approach to knowledge representation and is hence not limited by the above.

## **1.2 Objectives**

Research is needed to find techniques to process a standard's knowledge. This particular research effort is not only to process a single standard but also to encode many standards in a module called "Common Interface of Design Standards", recognizing fact that an engineer may need to refer many standards during a particular design. The concept of "common interface" will assist the engineer to get familiarised with new standards by comparing the output of the older version or the familiar standard's version. The available high level software could be used to develop a flexible expert system on design standards which could assist the design engineer effectively. Implementation has been carried out however only for one standard, namely BS 8110.

The following main objectives were focused on to be achieved as the outcome of the research;

- Develop a framework to incorporate many standards in one module.
- Develop a structure in software to model and process a standard's knowledge as an object oriented model.

The framework and software were to have the following key features.

- Data entry for analysis and design to be in one operation. Thus data transfer and storage will be handled from one module. That is data entry, data transfer to external analysis software, retrieving the analysis output, conformance

checking of a particular element according to a selected standard and the stress state and storage of input and output data will be carried out by one module.

- A flexible system where revisions can be adopted with less effort. Thus the programme is to be developed without hard coding.
- Object oriented techniques to be use to develop the software. This technique is a very popular programming technique presently adopted by software engineers.
- Processing of data to simulate the procedure followed by a practicing engineer.
- Representation of standards without distorting the standard, i.e. to represent text and knowledge types as represented in the standards. Thus the representation of code clauses will be carried out as stated in the code and the tables in the same format. This will give the flexibility to the system in the event of code revisions. The other advantage is that the debugging and establishing the connectivity will be very efficient.

### **1.3 Methodology**

The scope of the research was to develop a common interface for reinforced concrete design standards, for conformance checking of structural elements, using Object Oriented Programming techniques. A literature review was done to investigate the current research work on the subject areas such as standards processing and expert systems. Several reinforced concrete standards were reviewed to understand their structure and design philosophy. Object oriented techniques were used for the implementation, using the object oriented programming shell “KAPPA”.

#### **1.3.1 Thesis Structure**

The Thesis consists of seven chapters, the first being this Introduction. The rest of the thesis is as follows:-

Chapter 2: Representation of Design standards - This chapter examines the nature and structuring of knowledge in standards. The discussion leads to the possibilities of standards representation as expert systems. An overview of existing standards processing techniques such as Predicate Logic, Decision Tables, Production Systems, Frames and Semantic Networks is given. Finally the Object Oriented representation of

Standards is discussed, the technique adopted to develop the “Common Interface for Design Standards”.

Chapter 3: Comparison of Design Standards - Since the research is to develop techniques to model a common interface of design standards, it is important to compare the structure and the philosophy adopted by the standards. This chapter gives various comparisons of reinforced concrete design standards.

Chapter 4: Common interface concept and implementation - This chapter gives an overview of the concept of a common interface, highlighting the three main models of the Common Interface of Design Standards (COIDS), namely the product model, standards model and the interaction model. The respective hierarchies of each model are discussed in detail. The main object classes, sub-classes, instances and their tasks and relationship with respect to the common interface are discussed. The structure of each object, i.e. the slots and methods are also discussed in detail.

Chapter 5: KAPPA application development Software: - This chapter gives an overview of the KAPPA application development environment, Object Oriented programming concepts and introductions to inferencing techniques such as forward chaining and backward chaining which have been applied in the development of COIDS. This chapter demonstrates the COIDS dynamic execution process of instructions by message passing from one object to another based on the user input.

Chapter 6: Typical COIDS sessions - This chapter gives an overview of a typical COIDS session in the data entry mode and the checking mode, which demonstrates the simulation of a practicing engineer.

Chapter 7: Conclusions - This chapter presents the conclusions drawn from the research. Suggestions for changes and extensions that could form part of future research are also made.

## **CHAPTER 2**

### **DESIGN CODES AS EXPERT SYSTEMS**

#### **2.1 Introduction**

This chapter will discuss the basic structure of design standards and the techniques that are used to represent them as expert systems, including finally, the author's approach of representing standards as Object Oriented representation. This chapter is based on the excellent unpublished literature review by Kodagoda (1997) and the well written thesis by Neilson (1997).

#### **2.2 Design Standards**

Design standards are regulations or provisions that state requirements which have to be satisfied to ensure safe and serviceable performance of certain systems during a specific time period. The two basic modes in which standards are used by practising engineers are;

- Designing systems / components
- Checking that a previously configured system or component conforms to a standard.

When designing a system or component within the scope of a standard, an experienced engineer will select those requirements within the scope of standard that he or she judges will govern the design (based on the behaviours addressed by those requirements) and focus only on those requirements when synthesising the design. After synthesising the design for this subset of requirements, an engineer determines whether the current design also meets the applicable requirements which are not yet considered (Garrett, 1990). This is true for checking of a system or component.

### 2.3 Aim of building standards

It is important to understand the basic aims of standards, in order to achieve the maximum benefit from processing standards. Generally we could outline the following aims;

- The provisions of the standards are to ensure that all the criteria relevant to safety, serviceability and durability considerations are met. Thus it ensures an acceptable probability that the structure or part of it will not attain any specific limit state during its expected life.
- To organise and express the regulations in such a manner that they are logically consistent and complexity is minimised.
- To ensure that the benefits of the regulations are worth the cost and effort of implementing them (Blackmore, 1989).

### 2.4 Properties of standards

Required properties of standards have been defined by the NBS/CMU group (Stahl et al., 1983). They are as follows;

#### a. Individual provisions should be:

- **Unique** - provisions should give one and only one result for any given application.
- **Complete** - provisions should apply in any possible situation.
- **Correct** - the results of the application of the provision should be consistent with the intent of the standard.

#### b. Relations between provisions should be

- **Connected** - there should be explicit cross referencing of data items (variables) used within provisions.
- **Acyclic** - there should be no circular reasoning or requests for data items which represent logical loops.

**c. Organisation of the overall standard should be**

- **Complete** - the user should have some idea of the subjects and qualities covered by the standards.
- **Clear** - the provisions should be arranged in such a way that checking routines should be easily able to locate the provisions applicable to given queries.

## **2.5 Expert Systems**

Most engineers use computers for their calculation work and for drawing purposes using readymade software packages. These packages basically store and process data according to their predefined programming which is essentially executing a predefined task. Artificial Intelligence (AI) concepts have moved computers from functioning merely as data processors to functioning as knowledge processors.

Computers can now incorporate the knowledge of human experts to solve difficult problems. These systems are called expert systems. While a significant amount of knowledge and knowledge types can be stored in an expert system, the power comes from its ability to reason beyond the knowledge directly stored. This ability is called **inferencing**. There are two main methods of reasoning when using inference rules, namely forward chaining and backward chaining.

Forward chaining starts with the data available and uses the inference rules to derive more data until a desired goal is reached. An inference engine using forward chaining searches the inference rules until it finds one in which the “if” clause is known to be true. It then concludes the “then” clause and adds this information to its data. It would continue to do this until a goal is reached. The data available determines which inference rules are to be used.

Backward chaining starts with a list of goals and works backwards to see if there is data which will allow it to conclude any of these goals. An inference engine using backward chaining would search the inference rules until it finds one which has a “then” clause that matches a desired goal.

### 2.5.1 The function of an expert system

The following functions can be highlighted for an efficient expert system;

- The Expert system should have the ability to store and process knowledge. Knowledge may be represented in different types; generally it can be text, numerical values, algebraic expressions, Boolean expressions, table format or graphical format.
- The user's ability to query the system in order to get required information, for example a requirement such as the minimum beam width for fire resistance.
- Flexibility of the expert system is very important; this means the system should have the ability to grow with new knowledge and should be able to respond to new requirements.
- The expert system should be able to link to other software in order to access and store required data from data bases, and to handle the properties of other software such as data storage, algebraic function handling, graphic handling etc.
- The Expert system should have the ability to model the entire problem.

### 2.6 Model - Based Reasoning

Early expert systems were designed to produce computer solutions to problems that only human experts could solve. These system generally used "if – then" rules to store or represent the knowledge of the human experts. (Rules are pieces of knowledge that can be combined in various ways to solve a wide variety of problems.)

For example;

**If**

The building includes shear walls to resist wind

### **Then**

The building is a braced building.

Rules are useful to represent expert knowledge, but they express only the surface of the knowledge. The system does not understand what a shear wall is or its arrangement, how the shear wall resists wind and why shear walls are needed to brace the building, since the system does not have a model of how to brace buildings or how wind is resisted by building elements.

The drawbacks of a simple rule system are such that it will fail when faced with a new situation; on the other hand whereas the model of a shear wall or a braced building will not only include descriptive representation of what a shear wall or braced building is, but also include detailed methods of calculating procedures to evaluate strength of the shear wall, bracing methods of buildings, the ability to evaluate the necessity to provide shear walls and the adequacy of the number of shear walls to brace the structure. This kind of a module can be used for checking a building structure against wind loading.

Reasoning that incorporates a model or simulation of this kind is called model - based reasoning. Since Engineers are primarily concerned with the design and diagnosis of complex systems, it should come as no surprise that model-based reasoning is a very useful tool for assisting in engineering decision making (Garrett, 1990). While causal models can, in principle, be created using only rules, in practice it is more convenient to combine rule - based and method - based reasoning within a domain of structured objects.

The domain of a knowledge - based system is the part of the world with which the system is concerned. The domain of COIDS is building elements. The task of COIDS is designing and checking building elements. The rule-based approach focuses on the task, not on the domain.

The model - based approach focuses on the domain; thus we can use the same model to accomplish multiple tasks without using the rules alone. An Object Oriented Programming problem solving strategy may be applied for creating the model.

## **2.7 Design Standards as Expert Systems**

Design Standards and their regulations / provisions represent readily available knowledge compiled over a reasonably long period of time by domain experts and as a result, many of the knowledge acquisition problems could be regarded as already having been dealt with at a certain level.

Design codes correspond reasonably well to available knowledge representation formalisms, and the processing of provisions could be handled using inferencing mechanisms available in expert systems. The chaining of provisions can also be handled by tools. Thus design standards linked in COIDS can be regarded as a comprehensive expert system.

### **2.7.1 Standards processing**

The computerisation of building regulations are known as standards processing. Standards processing is carried out using expert system tools. We should be mindful not to distort the structure and properties of the standard during the encoding process.

The basic building blocks of standards comprise clauses /provisions. Provisions are collections of one or more rules which outline the procedures to evaluate and relate data items (variables) of the standards. A data item may be of any of the following types:

- Boolean, which can be evaluated to 'TRUE' or 'FALSE'.
- Numeric, which can be evaluated to a number.
- Multi -valued, which can be evaluated to more than one value.
- Singled - valued, which can only be evaluated to a unique value.

The data items in standards may need to be evaluated using algebraic or logical functions, or the lookup routines to be accessed from tables. This means that knowledge represented in standards is not homogeneous.

### 2.7.2 Classification of Clauses

For the purpose of standards processing, all the clauses could generally be classified in the following three categories;

- **Definition Clauses**; these Clauses will define the situation, for example CI 3.4.1.1 of BS8110 (1997)/ Part 1 defines the difference between a shallow beam and deep beam.
- **Application Clauses**; these Clauses specify the requirements in order to undertake a particular task. For example, CI 3.4.3 of BS8110 (1997) underlines the requirements for the application of table 3.6.
- **Performance Clauses**; these Clauses state the procedures for evaluation of data items. For example, CI 3.4.1.2 to CI 3.4.1.4 of BS8110 (1997) states the relevant procedures to evaluate the data item named “Effective – span”.

### 2.7.3 Parts of a Clause

The purpose of any clause is to establish certain criteria based on the satisfaction of other requirements (i.e. criteria). Thus clauses consist of one or both of the following two types of criteria:

- Applicability criteria
- Performance criteria

The applicability criteria of a clause are the required conditions to be satisfied in order that the performance criteria may be evaluated. The applicability criteria can be viewed as the **If** parts of a rule and the performance criteria as the **Then** part a rule. Thus clauses that could be represented as rules should consist of both criteria.

## 2.8 Standards processing techniques

There are many programming techniques that can be adopted for standards processing. These techniques will be discussed in detail later in this chapter. They are as follows;

- **Predicate Logic:** A formal systematic means for representing atomic variables and relations between them. The language Prolog supports implementation of this technique.
- **Decision Tables:** This consists of a set of conditions, actions and rules. This technique allows a set of rules concerned with the name of the object to be processed simultaneously.
- **Production System:** They are rule base systems containing condition-action rules called productions. A majority of the early Expert Systems were developed based on this technique. They have been popular since they are usually very readable and each rule is independent of the others, making modification easy.
- **Frames:** This consists of a hierarchy of Frames, each Frame containing a series of slots which describe that Frame. Frames can inherit information from other Frames somewhat like a semantic network. These have been used to model complex data.
- **Semantic Networks:** Represents objects and their relations by the use of nodes and links. The relationships between objects are in the form of a general graph. This involves building up a hierarchy of data items. Inheritance can be used for this purpose.
- **Object Oriented Representation:** This is the technique that has been used for developing COIDS. The main properties of this technique are Abstract data Types, Inheritance, Encapsulation, Polymorphism and Message Passing. The main benefit of this methodology is in software maintenance and reusability. In modelling a

system the main emphasis is placed on development of a hierarchy of objects. An object can be represented as a physical object or an abstract item. The behaviour of each object is coded in the object itself. Object oriented modelling has been used successfully to model complex systems and handle simulations. Most of the present high level programmes / languages are developed using object oriented techniques.

### **2.8.1 Criteria for selecting a suitable technique and software for standards processing**

In selecting a suitable knowledge representation technique and software to process design standards, the following considerations apply;

- The technique should be efficient and convenient to represent the standards
- The software should be able to handle numerical calculations, logical arguments, tables, and other linked software such as analysis software
- The techniques should be easily accommodate any changes to the standards, since the standards are updated frequently
- The chosen technique should possess the ability to represent both the element and the element assemblies of a structure

### **2.8.2 Predicate Logic**

The computer language Prolog is based on first order predicate logic. Here one stores data as predicates which are actually facts. It is possible to have rules where the conditions (antecedent) can be combined with AND, OR, NOT. If the rule is satisfied then the given consequent is done.

Predicate logic has been used in areas like theorem proving. Rosenman and Gero (1985) have used Prolog to develop an Expert System which represents a building standard. They have used both rules and facts to build up their knowledge base. It is possible to interactively engage in a session with the knowledge base to query about a specific clause in the code. This system also demonstrates how it is possible to query

the knowledge base, HOW it has arrived at a solution, or ask WHY a certain question is being asked. These features are some of the essential features found in rule based systems.

Fig. 2.1 demonstrates how a Table taken from a design code should be implemented as facts. The predicate `get_cover_beamsSimplySupported` is an example of a rule, that determines the cover required for a desired fire resistance period. If we query the Knowledge base with `get_cover_beamsSimplySupported (2.0, X)` then X would contain 40. If the query is altered to `get_cover_beamsSimplySupported (X, 60)` then X would contain 3.0.

```

table_4_3 (0.5, 20, 20, 20, 20, 20, 20, 20).
table_4_3 (1.0, 20, 20, 20, 20, 20, 20, 20).
table_4_3 (1.5, 20, 20, 25, 20, 35, 20, 20).
table_4_3 (2.0, 40, 30, 35, 25, 45, 35, 25).
table_4_3 (3.0, 60, 40, 45, 35, 55, 45, 25).
table_4_3 (4.0, 70, 50, 55, 45, 65, 55, 25).

get_cover_beamsSimplySupported (Fire, Cover):-
    table_4_3 (Fire, Cover, . . . . . ).

```

**Fig. 2.1: Prolog Implementation of a Table**

The symbol ‘:-’ represents **if**, and the symbol ‘,’ represents **and** in Fig. 2.2. The representation of a rule in Prolog is similar to the use of a Procedure in a conventional programming language.

Kumar and Topping (1989) have illustrated the utilization of Prolog to represent a Steel Design Code. They have proposed a technique in which they represent the entire code as facts. They have divided the facts stored in a code into three categories. The clauses are stored in such a way that the relevant rules are stored as facts separately. These are linked together by the clause numbers.

Only the first two requirements are satisfied in implementing a design code. The knowledge representation closely follows how it is actually implemented in the written clause itself, although this may not necessarily be the best method for computer implementation.

$\text{beam\_steel\_tension\_area}(BM, BEW, BED, FCK, FCY, XOD, As):-$ $BG\_MU = BM / (BEW * BED * FCK),$ $BG\_W = 0.652 - \text{sqrt}(0.425 - 1.5 * BG\_MU),$ $XOD = 1.918 * BG\_W,$ $As = BG\_W * BEW * BED * FCK / FCY$
--

**Fig. 2.2: Prolog Implementation of Calculation of Tension Steel area in a Beam**

### 2.8.3 Decision Tables

A Decision table is composed of sets of conditions, actions and rules. This provides a compact form of handling different actions as a result of a set of rules. Decision tables have been used in several projects concerning regulations and codes (Harris and Fenves, 1980). Fig. 2.3 shows an implementation of  $(x/d)_{lim}$  ratio which is needed in the calculation of compression steel reinforcement area in the EC2 concrete code.

$(x/d)_{lim}$		
$f_{ck} \leq C35/45$	T	F
$(x/d)_{lim} = \frac{(\delta - 0.44)}{1.25}$	X	
$(x/d)_{lim} = \frac{(\delta - 0.56)}{1.25}$		X

**Fig. 2.3: Calculation of  $(x/d)_{lim}$  using a decision table**

Fig 2.4 shows how multiple conditions can be handled by a decision table. The calculation of longitudinal shear spacing is done according to the EC2 concrete code.

Garret and Hakim (1992) have described in detail the disadvantages of Decision Tables. The two main disadvantages they observe are;

- a) Lack of formal model to implement the design code objects (i.e., grouping rules together is not possible)
- b) Lack of methods to handle other types of data evaluation (e.g. tables)

This technique would be unsuitable to handle a design standard with many requirements.

Is			
$V_{sd} \leq (1/5) V_{Rd2}$	T	F	F
$(1/5) V_{Rd2} \leq V_{sd} \leq (1/3) V_{Rd2}$	F	T	F
$V_{sd} > (2/3) V_{Rd2}$	F	F	T
$Is = \min(0.8d, 300)$	X		
$Is = \min(0.6d, 300)$		X	
$Is = \min(0.3d, 300)$			X

**Fig. 2.4: Calculation of Longitudinal Spacing for shear using a decision table**

### 2.8.3 Production Systems

Production Systems are the most popular technique used to develop expert systems. This consists of a collection of rules in the form of an IF part and a THEN part. Each rule is independent from the other. The main disadvantage of this representation is that when the knowledge-base becomes larger there are inefficiencies (Rossnman and

Gero 1985). This methodology has been successfully applied to implement Diagnostic type of Expert System.

Figure 2.5 shows the Production System version of the Calculation of Tensile Steel area in a beam. Rule based systems are very readable. These are not ideally suited for numerical computations. In rule based systems it's preferable to have only one consequent (fact given after THEN). This is to ensure that the inference mechanism (Backtracking) would be implemented properly.

Figure 2.6 illustrates how backtracking works. This technique is also used in Predicate Logic based methods. The stack is a storage location used by the inference engine. One feature of a Stack is that the first item popped out from the stack is the last item that is pushed in. (Rule 2,1) means that when Rule 2 was pushed into the Stack the inference engine evaluated Condition 1.

Production systems are similar to Decision Tables. Their use as Design Code representation becomes impractical due to the enormous amounts of data and rules which are required to be implemented in a design code.

```
Rule 1
IF element is a Beam
AND bending_moment is not unknown
AND beam_effective_width is not unknown
AND beam_effective_depth is not unknown
AND fck is not unknown THEN
beam_greek_mu = bending_moment / (beam_effective_width * beam_effective_depth^2 * fck)

Rule 2
IF beam_greek_mu is not unknown THEN
beam_greek_w = 0.652 - sqrt(0.425 - 1.5 * beam_greek_mu)

Rule 3
IF beam_greek_w is not unknown THEN
x_over_d = 1.918 * beam_greek_w

Rule 4
IF x_over_d is not unknown
AND fcy is not unknown THEN
As = beam_greek_w * beam_effective_width * beam_effective_depth * fck / fcy
```

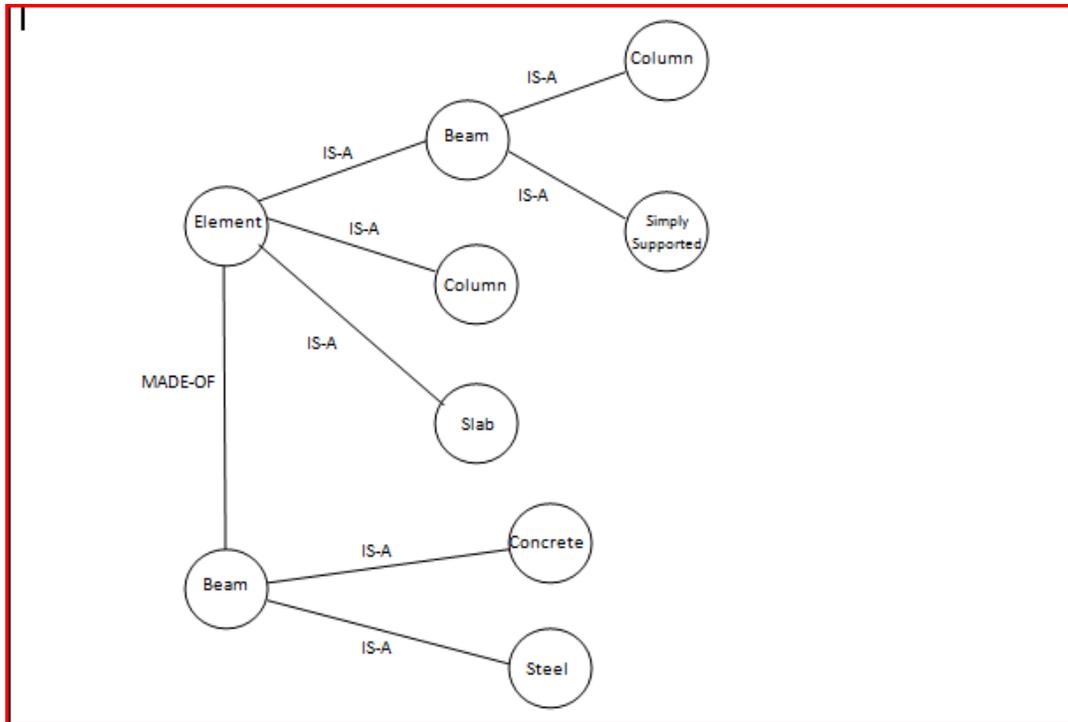
**Fig. 2.5: Production System Implementation of Calculation of Tensile Steel area in a Beam**

No	Goal	Description of what Inference engine does	Stack
1	Find As	Rule 4 can be used to obtain a value for As	Nil
2	Evaluate Rule 4	X_over_d value needed, if not available get value, store Rule 4 in the stack	(Rule 4,1)
3	Find x_over_d	Backtracking, Rule 3 contains a value for x_over_d	(Rule 4,1)
4	Evaluation Rule 3	beam_greek_w value needed, if not available get value, store Rule 3 in the stack	(Rule 3,1), (Rule 4,1)
5	Find beam_greek_w	Backtracking, Rule 2 contains a value for beam_greek_w	(Rule 3,1), (Rule 4,1)
6	Evaluate Rule 2	beam_greek_mu value needed, if not available get value, store Rule 2 in the stack	(Rule 2,1), (Rule 3,1), (Rule 4,1)
7	Find beam_greek_mu	Backtracking, Rule 1 contains a value for beam_greek_mu	(Rule 2,1), (Rule 3,1), (Rule 4,1)
8	Evaluate Rule1	element value needed, if not available and since no rules are available to get value, get value from user	(Rule 2,1), (Rule 3,1), (Rule 4,1)
9	Evaluate Rule 1	Bending moment value needed, if not available and since no rules are available to get value, get value from user	(Rule 2,1), (Rule 3,1), (Rule 4,1)
...	...	...	...
13	Calculate As	Rule 1 complete, Pop rule from stack, Backtrack to earlier rule (Rule 2)	(Rule 3,1), (Rule 4,1)
14	Calculate beam_greek_w	Rule 2 complete, Pop rule from stack, Backtrack to earlier rule (Rule 3)	(Rule 4,1)
15	Calculate x_over_d	Rule 3 complete, Pop rule from stack, Backtrack to earlier rule (Rule 4)	Nil
16	Find Fcy	Fcy value needed, if not available and since no rules are available to get value, get value from user	Nil
17	Calculate As	Goal finally satisfied	

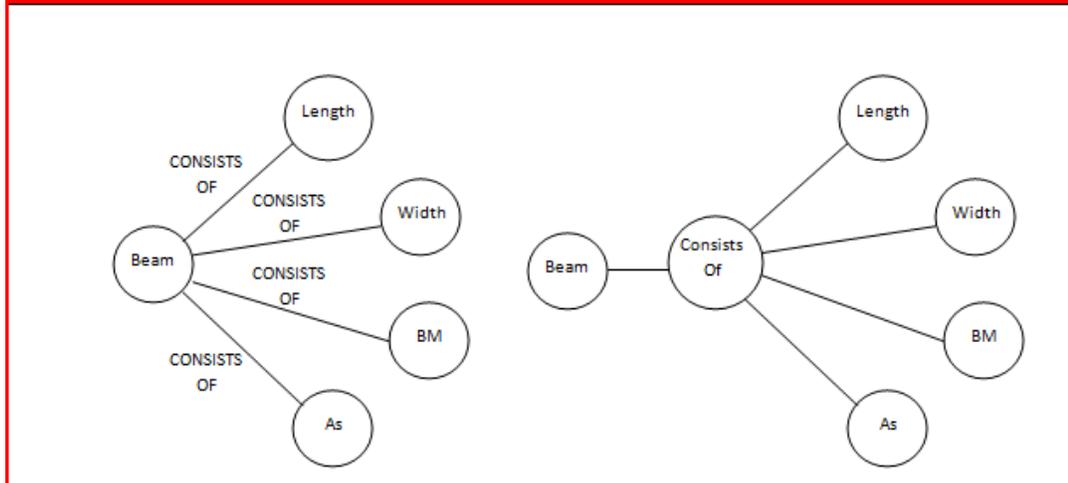
**Fig. 2.6: Details of Backtracking through a rules given in Fig. 2.5**

### 2.8.4 Semantic Networks

These are basically graphical descriptions of knowledge that show hierarchical relationships between objects. An object can be a physical entity like a beam or an abstract entity like shear force. These are usually referred to as nodes. Nodes can be connected together by arcs to represent relationships between them. The IS-A relationship is basically a class relationship. Semantic Networks are a good medium to represent complex relationships. Other types of relationships can also be defined. A complete detailed semantic network for an element hierarchy is shown in Fig 2.7. Adding details to one of the objects can be carried out as shown Fig. 2.8.



**Fig. 2.7:** Semantic network which shows the relationship of physical objects in a design standard



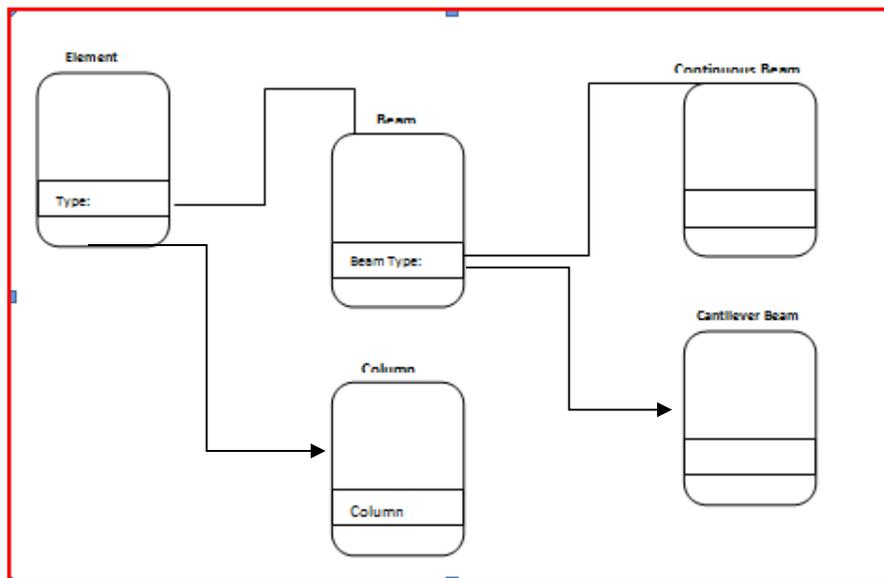
**Fig. 2.8:** Semantic network that shows details of one particular object

### 2.8.5 Frames

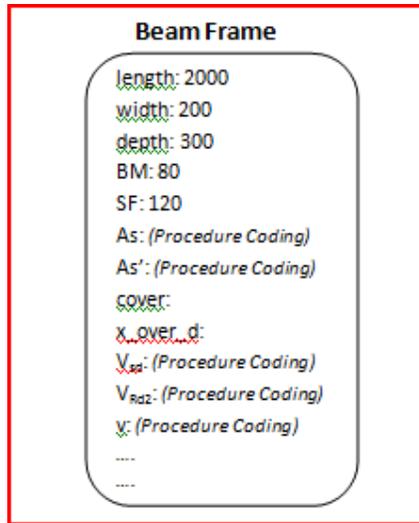
A frame contains a large chunk of knowledge about a particular object. A frame provides a means of organizing knowledge in slots that contain characteristics of that

object. Some slots can contain default values; some could contain a procedural attachment. It is possible to use a slot to link to another frame. It's possible to develop a hierarchy of objects (see Fig. 2.9). For example the continuous beam frame inherits all the properties of a general beam frame and a more general element.

Frames have been used to implement Expert Systems to assess damages caused to existing structures (Zhang and Yao, 1989). By attaching a procedure to a slot one can handle numerical calculations. Its ability to use a slot to link to another frame allows it to be used to even implement aggregates of elements. Frames provide a mechanism of storing all the relevant information of an object in one location which also includes calculations (see Fig. 2.10). This makes modifications easier. Frames satisfy all four conditions mentioned earlier and are a suitable technique to implement a design code. The principles adopted in this technique are very similar to the Object Oriented Programming techniques which have been used to develop COIDS.



**Fig. 2.9: Knowledge Representation in a hierarchy of frames that show inheritances**



**Fig. 2.10: Frame of a Beam**

### 2.8.6 Object Oriented Programming Technique

Object Oriented Programming (OOP) is a relatively new way of organising programming code and data. Its underlying concepts are Data Abstraction, Inheritance, Encapsulation and Polymorphism. These concepts have been around for some time, for example in languages such as Simula67 and Smalltalk.

Object Oriented Programming is part of a long process of improving programmer productivity that has moved from standard programming to structured programming to OOPs. For years, all professional programmers have reused the programming code. A very common approach to a new programming assignment is to copy an existing program and modify it to solve the new problem. This approach has both benefits and problems. The major benefit is that you start with a body of working programming code. This is particularly useful in areas like Windows programming, where applications inevitably require the same basic programming code for things like handling windows, menus, and other common elements. However, there are also drawbacks, most notably that every time you change your code, you risk introducing a new bug. As programmers grew in experience and knowledge, the problems of dealing with reuse of traditional procedural programming code were addressed by following specific rules, called **structured programming**, for creating blocks of code. This was an improvement, because it made code easier to understand and easier to debug, but still did not accomplish the ultimate goals of allowing you to add to

existing programming code easily and reliably. Object Oriented Programming evolved to solve this problem.

In conventional Procedure Oriented Programming in a language such as C, you view a problem as a sequence of things to do. You organise the related data items into C structures and write the necessary functions (procedures) to manipulate data and, in the process, complete the sequence of tasks that solve your problem. Although the data may be organised into structures, the primary focus is on the functions. Each C function transforms data in some way. For example, you may have a function that calculates the average value of a set of numbers, another that computes the square root, and one that prints a string; C-function libraries are implemented in this manner.

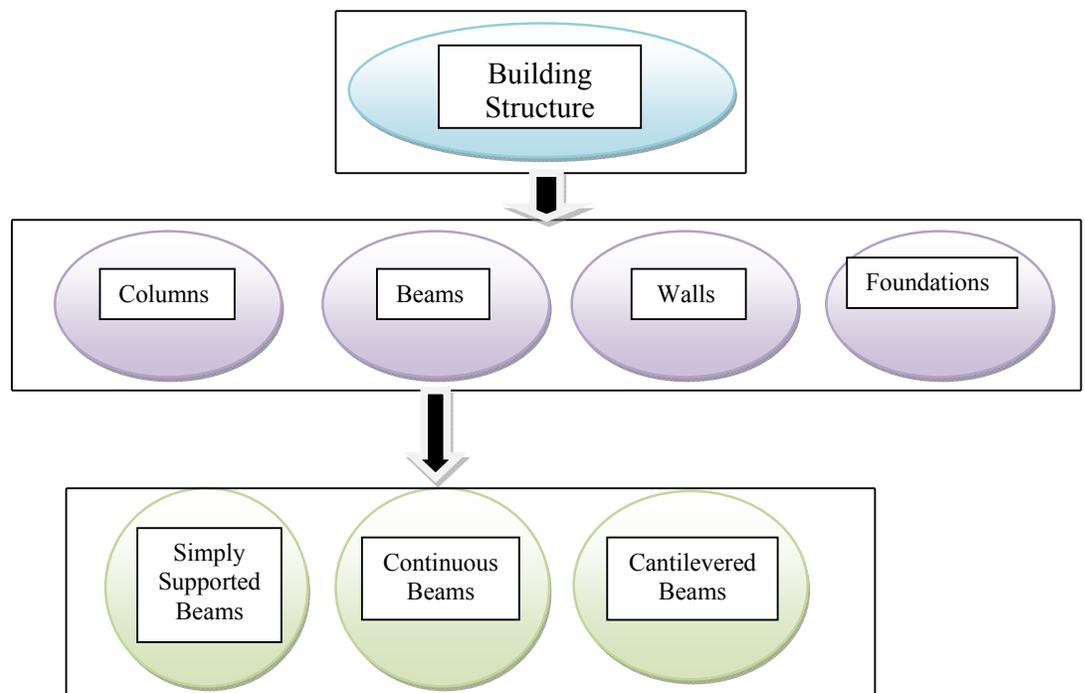
OOP is more about program organisation than programming code techniques and it has nothing to do with any programming language. A programming language that supports OOP makes it easier to implement OOP techniques. Object oriented programming looks at a problem as a collection of data, rather than collections of processes or functions. When you are working with OOP, you think first about what data is required, rather than on what needs to happen to the data.

Object oriented programming enables you to remain close to the conceptual, high-level model of the real-world problem which you are trying to solve. You can take the advantage of the modularity of objects and implement the program in relatively independent units that are easier to maintain and extend. You can also share programming code through inheritance. The entire objective of OOP is in fact the defining of objects in an object-oriented manner that is in a unit that combines both data and functionality together.

There are four main features of OOP, namely, Data Abstraction, Inheritance, Encapsulation and Polymorphism;

- **Data Abstraction:** Fundamentally, to understand a complex system of data and its behaviour, macro level perception of the system is very essential. For example a building structure consists of beam, columns, walls and foundations. If we further observe in detail, there are simply supported beams, continuous

beams and cantilever beams (see fig. 2.11). Data abstraction is to tie data and functions together, which effectively defines a new data type with its own set of operations. Such a data type is called an abstract data type (ADT), also referred to as a “Class”. Probably the most difficult part of OOP is organising and understanding the object classes that make up your application. Defining the objects and setting up the relationship between them is a new task for the programmer. The novelty of this approach makes it harder initially than designing a traditional application.

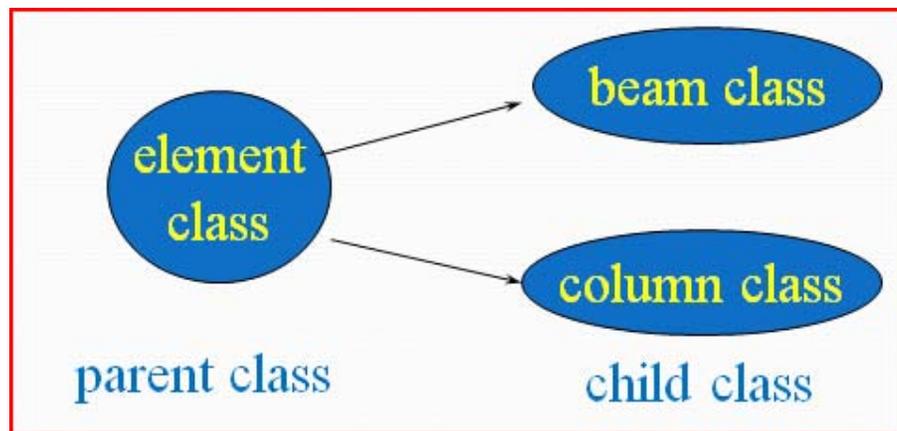


**Fig. 2.11: Data Abstraction levels of Building Structures**

- **Inheritance:** Real - world objects do not exist in isolation. Each object is related to other objects. In fact, we can describe a new kind of object by pointing out how the new object's characteristics and behaviour differ from that of a class of objects that already exists. This option of defining a new object in terms of an old one is an integral part of OOP. The term inheritance is used for this concept, because we can think of one class of objects inheriting the data and behaviour from another. Inheritance imposes a hierarchical relationship among classes in which a **child**

**class (derived class)** inherits from its **parent class (base class)** - see Fig. 2.12. For example, Beams class is derived from Element Data class. Thus it inherits all the data and functions (methods) from the base class Element Data class.

- If an object inherits from only one parent object, it is known as **single inheritance** (see Fig. 2.13). Real-world objects often exhibit characteristics that they inherit from more than one object. For instance a one way spanning slab may exhibit both beam and slab characteristics. This example illustrates **multiple inheritances** (see Fig. 2.13), the idea that a class can be derived from more than one base class. The hierarchical structure generated by the single inheritance is simple where as in multiple inheritances the hierarchical structure is much more complicated. Thus single inheritance is more popular among the programmers as the programme debugging is much easier. An individual representation of a class is known as an **Instance** (see Fig. 2.14). Thus all instances of a class will respond to the same instructions and perform in a similar manner.



**Fig. 2.12: Object oriented representation of Parent Class and the Child Class**

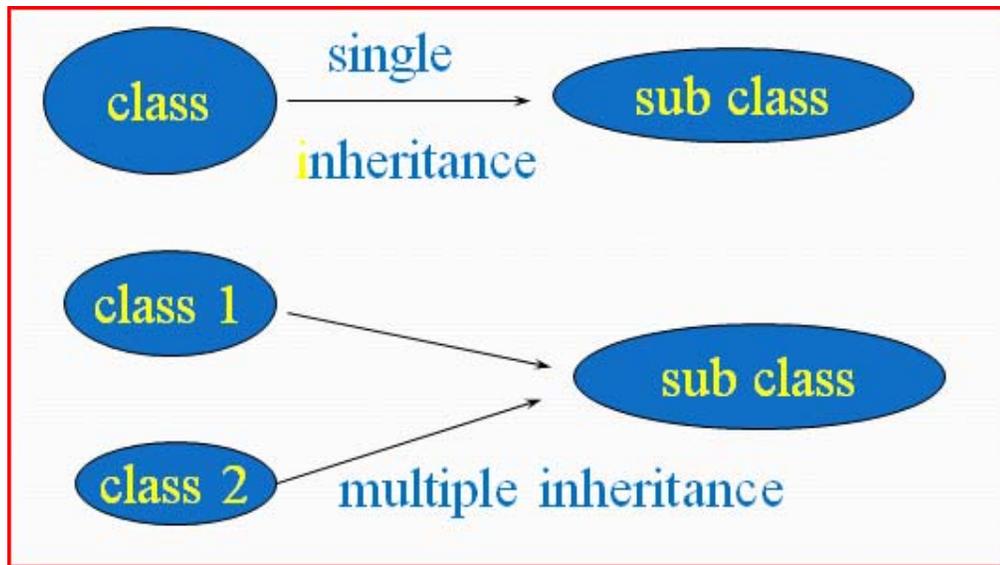


Fig. 2.13: Object oriented representation of single inheritance and the multiple inheritances

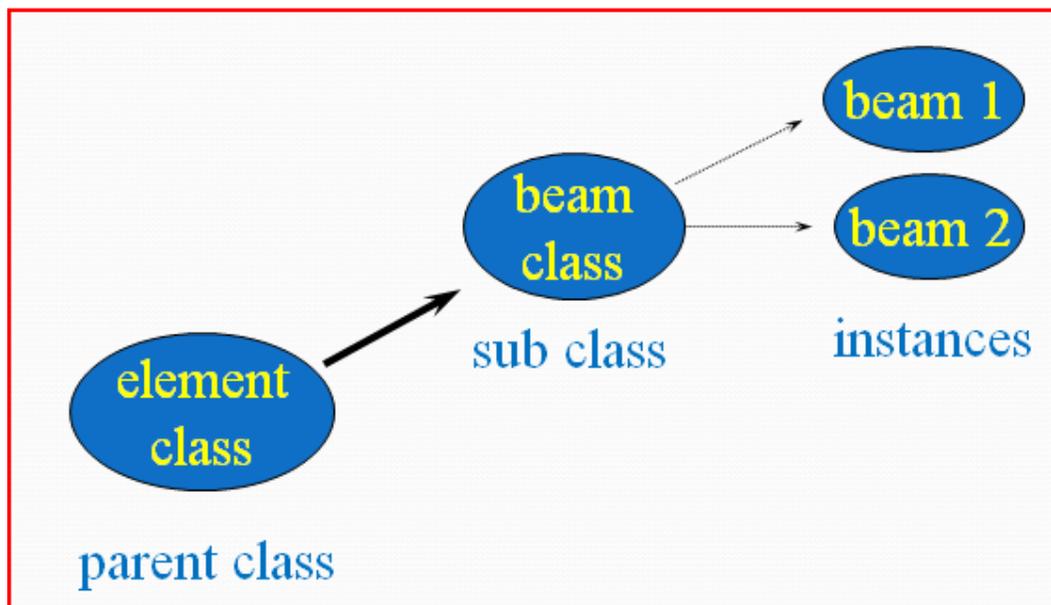


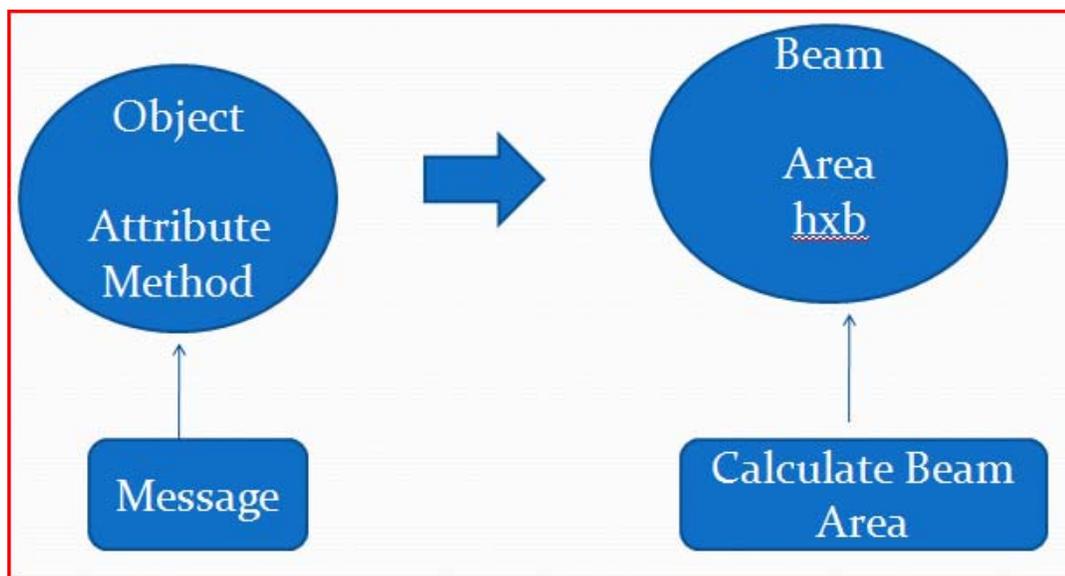
Fig. 2.14: Object oriented representation of object instances

- **Encapsulation:** Once we have determined the type of object in a class, we need to define its components or **slot values**, and how it is accessed. One of the most powerful and important features of OOP is that the access to a class is strictly regulated. The user of a class does not need to know details of how the class is

organised internally; all access to the class data is through a series of external function calls (**messages**), or internal methods. The process of binding class organisation so that it is not directly accessible to the user is called **encapsulation** (see Fig. 2.15)

Encapsulation has three important features:

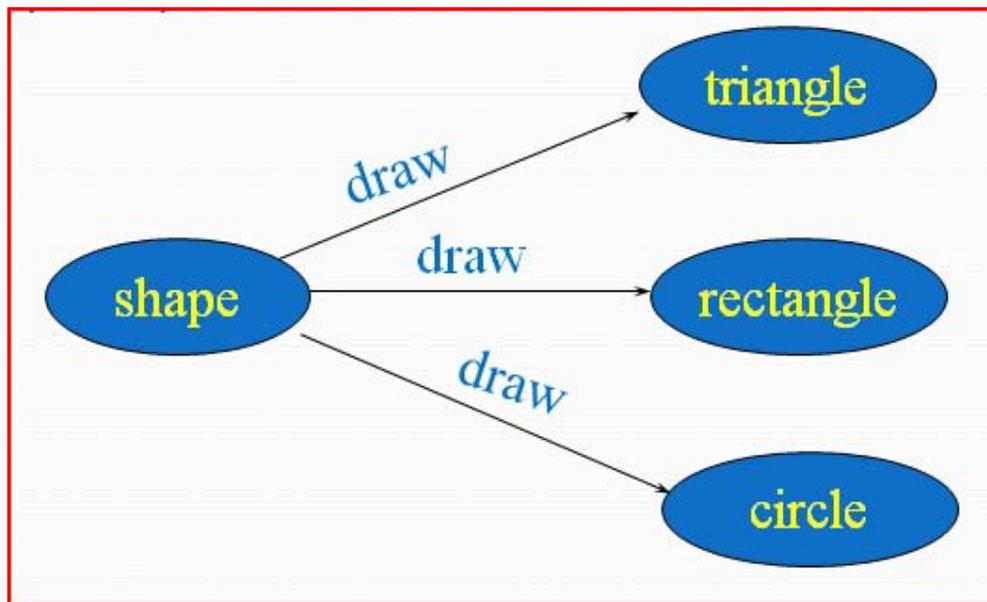
1. It provides a clear boundary that defines and protects all of an object's internal structure.
2. It defines an interface that describes and controls how other users work with an object.
3. It provides a protected, internal implementation of the object's behaviour and structure of the class itself.



**Fig. 2.15: Object Orientated representation of Objects, which includes both slot value (attribute) and the method to evaluate the attribute (Encapsulation)**

- **Polymorphism:** In a literal sense, “polymorphism” means the quality of having more than one form. In the context of OOP, polymorphism refers to the fact that a single operation can have different behaviour in different objects. That is, different objects react differently to the same message. For example,

suppose a number of geometrical shapes all respond to the same message, draw. Each object reacts to this message by displaying its shape on a display screen (see Fig. 2.16). The actual mechanism for displaying the object differs from one shape to another, but all shapes perform this task in response to the same message. The idea of polymorphism is that subclasses can have different functions that respond to the same message as base class, but they can perform different functions. Polymorphism helps by simplifying the syntax of performing the same operation on a collection of objects.



**Fig. 2.16: Object oriented representation of concept polymorphism. A single message to carryout multiple task**

## **CHAPTER 3**

### **COMPARISON OF DESIGN STANDARDS**

#### **3.1 Introduction**

Common interface for design standards (COIDS) is an attempt to accommodate several design standards in a single model. Study of several reinforced concrete standards indicates there are similarities and dissimilarities between the standards. In order to address this aspect we ask the question, “Do different design standards perceive the basic reinforced concrete frame differently?” The study also attempts to understand the structure of the standards, design philosophy and the design approach with respect to element design. The author has reviewed several reinforced concrete design standards namely, British Standard BS8110 (1997), European Standard EC2 (1992), American Standard ACI 318 (2008), Australian Standard AS 3600 (1988), German Standard DIN 1045 (1978) and Indian Standard IS456 (2000). In general all standards follow the same design philosophy and approach to design of elements, i.e. the limit state philosophy. However there are many differences when we examine design standards at a micro scale. All these similarities and differences should be taken into account when formulating the common interface of design standards.

The main discussion of this chapter will be of BS 8110 (1997) and EC2 (1992); however the discussion will also highlight aspects of other standards in tabular format (See. Table 3.1).

#### **3.2 The structure of design standards**

Design standards are structured as element types, stress states (phenomena) or a combination of both types. For example, British Standard BS 8110 (1987) , under Section 3 (see Fig. 3.1), “Design and detailing of reinforced concrete” deals with structural elements design such as beams, slabs , columns, walls, staircases and bases. In Chapter 6 (see Fig. 3.2) of European Standard, Euro Code 2 (BS EN 1992-1-1:2004), “Ultimate Limit States”, sub sections are divided as bending with or without axial force, shear, torsion, punching, design with strut and tie models etc... Thus EC2 (1992) has by and large arranged its chapters on the basis of stress states (phenomena)

whereas BS8110 (1997) has been structured based on element type (Narayanan, 1994). Eurocode 2 does not contain derived formulae or specific guidance on determining moments and shear forces. This has arisen because it has been European practice to give principles in the codes and for the detailed applications to be presented in other sources such as textbooks (Moss & Webster, 2004).

<b>Section 3. Design and detailing: reinforced concrete</b>	
3.1	Design basis and strength of materials 15
3.2	Structures and structural frames 18
3.3	Concrete cover to reinforcement 21
3.4	Beams 26
3.5	Solid slabs supported by beams or walls 37
3.6	Ribbed slabs (with solid or hollow blocks or voids) 47
3.7	Flat slabs 50
3.8	Columns 65
3.9	Walls 75
3.10	Staircases 80
3.11	Bases 81
3.12	Considerations affecting design details 83

**Fig. 3.1: Part of the Contents (Section 3) of BS8110 Part 1 (1997) structured based on element type**

EN 1992-1-1:2004 (E)	
6.	Ultimate limit states (ULS)
6.1	Bending with or without axial force
6.2	Shear
6.2.1	General verification procedure
6.2.2	Members not requiring design shear reinforcement
6.2.3	Members requiring design shear reinforcement
6.2.4	Shear between web and flanges of T-sections
6.2.5	Shear at the interface between concretes cast at different times
6.3	Torsion
6.3.1	General
6.3.2	Design procedure
6.3.3	Warping torsion
6.4	Punching
6.4.1	General
6.4.2	Load distribution and basic control perimeter
6.4.3	Punching shear calculation
6.4.4	Punching shear resistance of slabs and column bases without shear reinforcement
6.4.5	Punching shear resistance of slabs and column bases with shear reinforcement
6.5	Design with strut and tie models
6.5.1	General
6.5.2	Struts
6.5.3	Ties
6.5.4	Nodes

**Fig. 3.2: Part of the Contents (Section 6) of Euro Code 2 (BS EN 1992-1-1:2004) structured based on stress states**

**Table 3.1: Structure and Philosophy of Design Standards**

<b>Standard</b>	<b>BS8110 (1997)</b>	<b>EC2 (1992)</b>	<b>ACI318 (2008)</b>	<b>AS3600 (1988)</b>	<b>DIN1045 (1978)</b>	<b>IS456 (2000)</b>
Philosophy	Limit State	Limit State	Limit State	Limit State	Limit State	Limit State
Structure	Element Type	Stress States	Stress States	Element Type	Element Type	Element Type
Units	SI	SI	Imperial	SI	SI	SI

### 3.3 Basis of Design

The basis of design of different design standards may differ depending on scope and the philosophy, for example, the Principles in EC2 Part 1 are meant to be applicable to all structures, therefore the character of the code is more general than it would need to be were it to be applied to buildings only. On the other hand, BS8110 is basically applicable to buildings. Thus it is able to provide information directly and in prescriptive form (Narayana, 1994). Thus encoding the BS8110 provisions in a computer module is easier than encoding provisions in EC2. Both standards uses limit state philosophy, i.e the ultimate limit state and serviceability limit state of design. Macro scale observations are made since there are many differences in micro structure of the standards.

### 3.4 Loads, Load Combinations and Partial Safety Factors

- **Loads:** The magnitude of loads to be used is the characteristic value of loads. The magnitudes of the characteristic values recommended by different standards differ. Table 3.2 highlights differences in typical recommended loading for residential and office buildings.
- **Load combination and partial safety factors:** The basic approach to establishing the design load is similar in most standards. The load combinations are considered for the dominant loads and the most unfavourable effect which is to be used for design. The patterns recommended to be considered and the load combinations to be adopted by different standards differ (see. Table 3.3).

**Table 3.2: Recommended Live loads**

Standard	BS6399(1996)	EC1 (1991)	ASCE 7	AS1170.1(1988)	IS875[2](1986)
Loads	Characteristic Load	Characteristic Load	Characteristic Load	Characteristic Load	Characteristic Load
Residential	1.5 kN/m <sup>2</sup>	1.5-2.0 kN/m <sup>2</sup>	1.92 kN/m <sup>2</sup>	1.5 kN/m <sup>2</sup>	2.0 kN/m <sup>2</sup>
Office	2.5 kN/m <sup>2</sup>	2.0-3.0 kN/m <sup>2</sup>	2.4 kN/m <sup>2</sup>	3.0 kN/m <sup>2</sup>	2.5kN/m <sup>2</sup>

**Table 3.3: Recommended basic load safety factors**

Standard	BS8110 (1997)	EC2 (1992)	ACI318 (2008)	AS1170.1 (1988)	DIN1045 (1978)	IS456 (2000)
Dead Load	1.4	1.35	1.4	1.25	1.75 <sup>[1]</sup>	1.5
Imposed Load	1.6	1.5	1.7	1.5		1.5

<sup>[1]</sup> Overall safety factor of 1.75 is adopted by DIN standards

### 3.5 Materials Properties

Material strength limits and the partial safety factors defined by different standards differ (see Tables 3.4 and 3.5).

- Concrete:** Constituents of cement, minimum cement content for a particular grade, maximum w/c ratio and concrete strength is not generally agreed among standards. Design formulae of different standards are based on cylinder strength of concrete (for example ACI318) or cube strength of concrete (for example BS8110). EC2 allows benefits to be derived from using high strength concretes, which BS8110 does not. The maximum characteristic cylinder strength  $f_{ck}$  permitted in EC2 is  $90\text{N/mm}^2$ , which corresponds to characteristic cube strength of  $105\text{N/mm}^2$  (Moss and Webster, 2004). Concrete density assumed by the different codes also differ, for example, EC2 assumes a

densities of 25 kN/m<sup>3</sup> for reinforced concrete and BS8110 assumes a density of 24 kN/m<sup>3</sup> for reinforced concrete.

- **Reinforcement:** In general characteristic yield stress is used in design formulae. Different standards stipulate different steel grades, for example, BS8110 defines two steel grades, namely grade 460 for high yield steel and grade 250, whereas EC2 specifies only Grade 500 steel.

**Table 3.4: Recommended Material Properties**

Standard	BS8110 (1997)	EC2 (1992)	ACI318 (2008)	AS3600 (1988)	DIN1045 (1978)	IS456 (2000)
Concrete Strength is based on	Cube Strength	Cylinder Strength	Cylinder Strength	Cylinder Strength	Cube Strength <sup>[1]</sup>	Cube Strength
Yield Strength of R/F	250 N/mm <sup>2</sup> and 460N/mm <sup>2</sup>	500 kN/m <sup>2</sup>	275N/mm <sup>2</sup> 413 N/mm <sup>2</sup> and 482 N/mm <sup>2</sup> <sup>[2]</sup>	250 N/mm <sup>2</sup> 400 N/mm <sup>2</sup> and 450 N/mm <sup>2</sup>	220N/mm <sup>2</sup> /4 20 N/mm <sup>2</sup> and 500N/mm <sup>2</sup>	250N/mm <sup>2</sup> / 415 N/mm <sup>2</sup> 500N/mm <sup>2</sup>

<sup>[1]</sup> Side of a cube is 200mm in DIN standards where as other standards adopt a cube side length of 150mm

<sup>[2]</sup> ACI318 (2008) has assign a yield strength value of 60000 psi (413 N/mm<sup>2</sup>), but makes provision for the use of higher strengths provided the stress corresponds to strain of 0.35 percent. However ASTM A 82 has a specified a minimum yield strength of 70,000 psi (482 N/mm<sup>2</sup>)

**Table 3.5: Recommended Material safety factors**

Standard	BS8110 (1997)	EC2 (1992)	ACI318 (2008)	AS3600 (1988)	IS456 (2000)
Concrete	1.5	1.5	1.11-1.538	1.25-1.67 <sup>[1]</sup>	1.5
Steel	1.05	1.15			1.15

[1] Overall material safety factor has been applied based on the stress state

### 3.6 Physical geometry of structures

Although the structure is identical whatever code is used to check it, structural definitions may differ from code to code – e.g. effective length and flange width of a beam (see Fig 3.3, Fig 3.4 and Table 3.6).

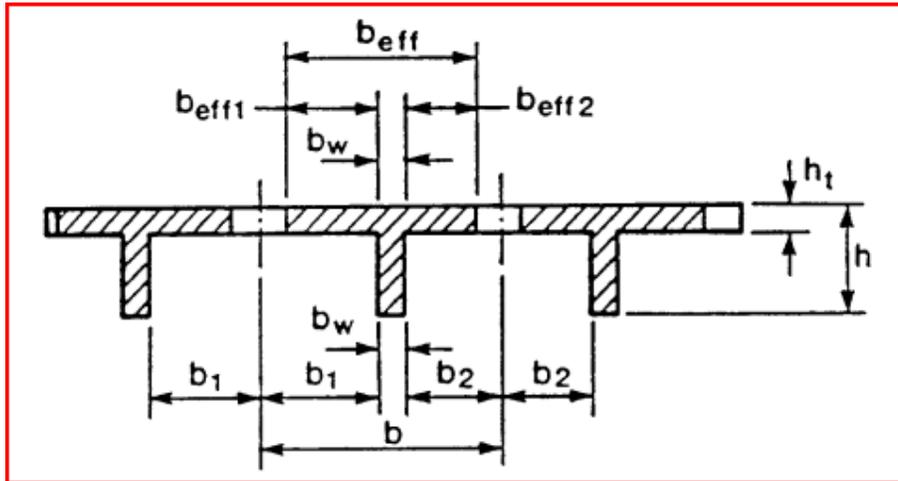


Fig. 3.3: Definition of Dimensions (Figure 2.2 of EC2)

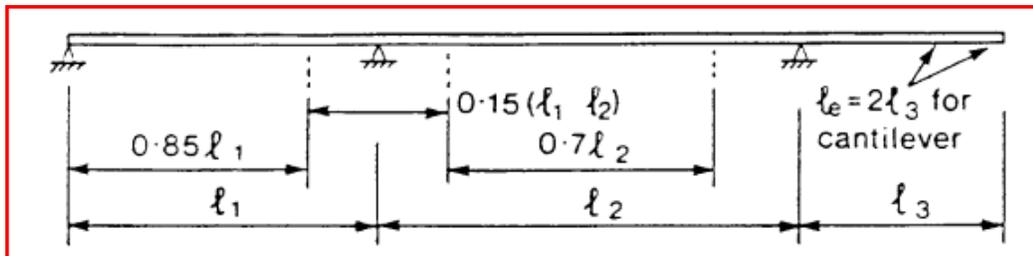


Fig. 3.4: Approximate effective span for calculation of effective breadth ratio (Figure 2.3 of EC2)

Table 3.6: Recommended equations to calculate the flange width of a beam

Standards	Recommended Flange width
BS8110(1997)	$b_{eff} = \frac{l_o}{5} + b_w$ , where $l_o$ = the distance between zero moments
EC2(2004)	$b_{eff} = \frac{l_o}{5} + b_w$ , where $l_o$ = the distance between zero moments
IS456(2000)	$b_{eff} = \frac{l_o}{6} + b_w + 6ht$ , where $l_o$ = the distance between zero moments

### 3.7 Other Key aspects of Design Standards

- **Durability:** Different design standards employ different approaches for defining the environmental condition of an element. Some standards provide general guidance, for example BS8110 (1997) Table 3.4 gives a general guide line on environmental conditions. Other standards define more specific guidelines on this aspect, for example, EC2 (1991) and AS3600 (1988). According to Narayanan (1994) BS8110 and EC2 have major disagreements. First, the principal parameters such as cement content, water/cement ratio, concrete strength and cover are not generally agreed on. Second, as noted above, the definitions of exposure conditions in the two documents are different and are entirely qualitative.
- **Analysis:** With respect to analysis there is a consensus about the commonly used method of analysis, for example elastic analysis with and without redistribution, plastic analysis and non-linear analysis methods. However different loading arrangements are prescribed (Ref. Table 3.7). According to Narayanan (1994), EC2 provides only the basic information required, whereas BS8110 gives considerably more detailed information. Thus bending moment coefficients for slabs and beams are given in BS8110, whereas EC2 expects the user to refer text books or manuals.
- **Design:** Plastic method of analysis is allowed in most standards but the recommended plastic stress diagrams differ (see fig. 3.5 and table 3.8). There has been some debate as to what is the most appropriate value to take for  $\alpha_{cc}$  in EC2. The recommended value in the code is 1.0 but it is likely that the UK National Annex will require a value of 0.85 to be used. The parameter  $\eta$  has been introduced into EC2 and in combination with modification of the value for  $\lambda$  has the effect of reducing the allowable concrete force for higher strength concrete grades above C50/60 (Moss and Webster, 2004).
- **Symbols:** Symbols too differ from standard to standard. This difference also should be considered when creating the common interface for design standards.

Table 3.7: Recommended Pattern loading in BS8110 and EC2

<p>BS8110</p>	
<p>EC2</p>	

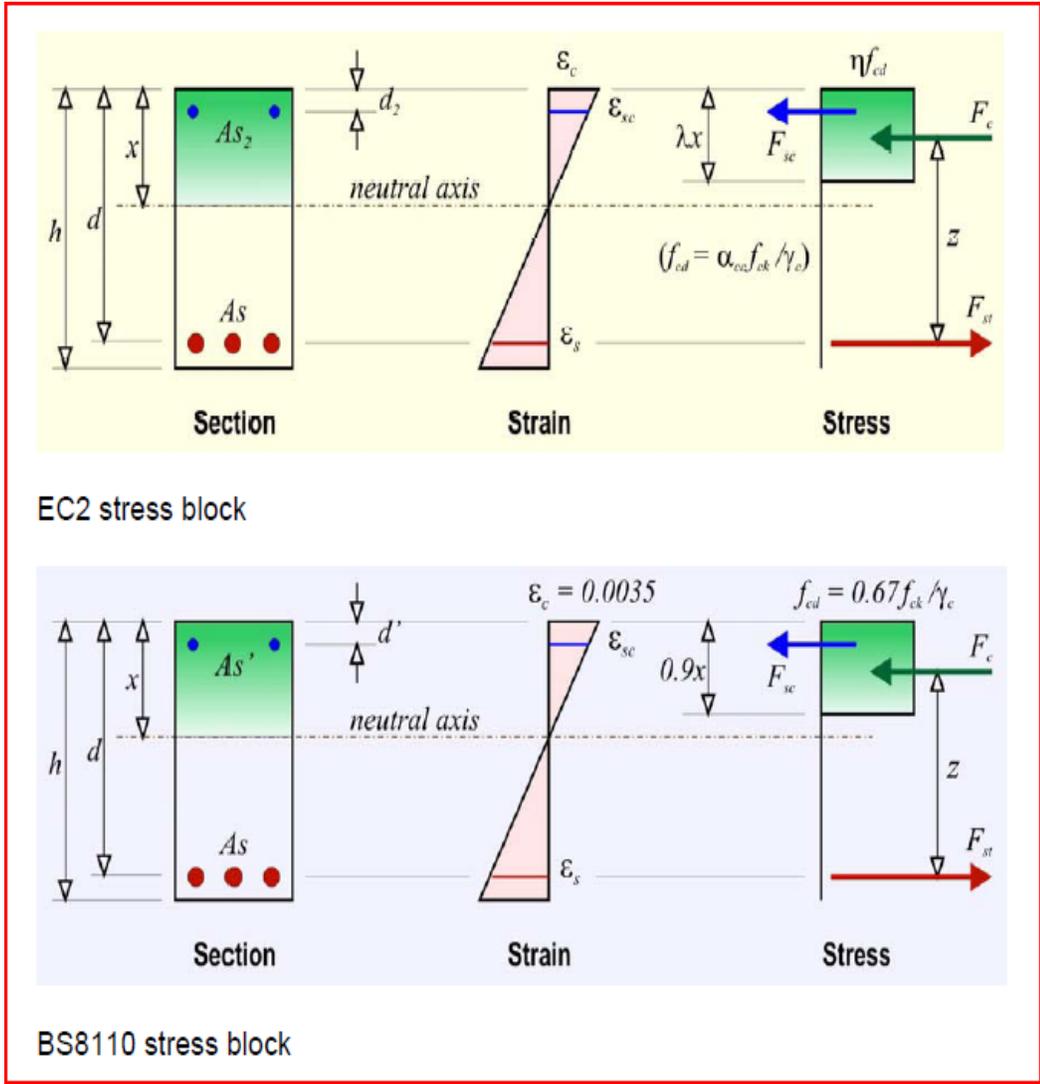
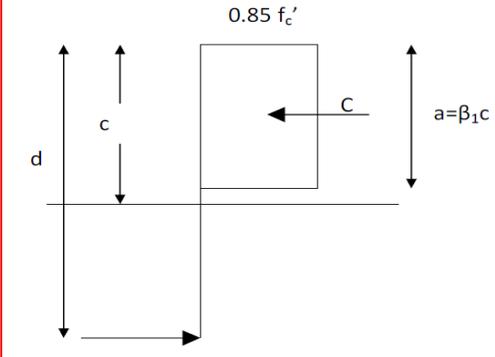
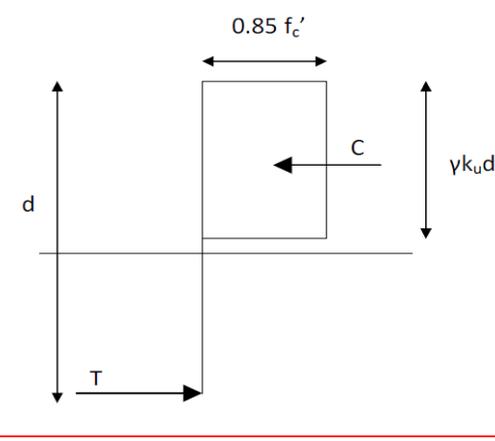
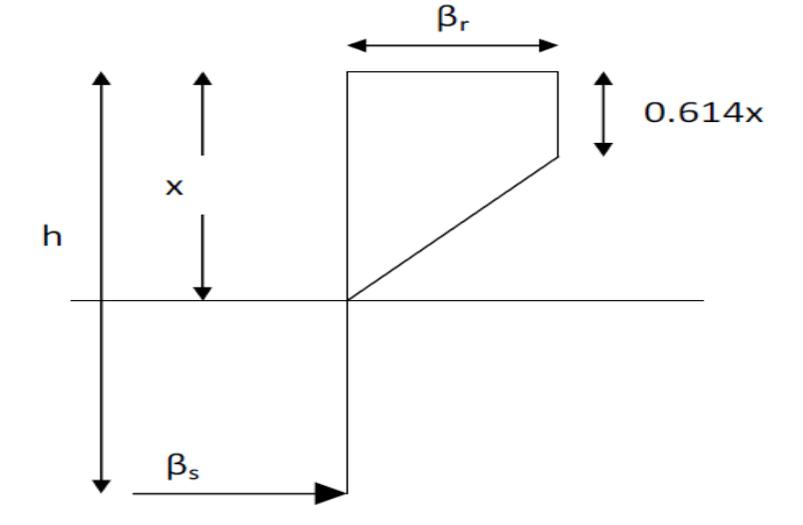


Fig. 3.5: Stress block used in EC2 is compared with that in BS8110 (Moss and Webster, 2004)

**Table 3.8: Recommended stress block diagrams by other standards**

Standards	Recommended plastic stress diagram
<p><b>ACI318</b></p>	 <p style="text-align: center;"><math>0.85 f_c'</math></p> <p style="text-align: right;"><math>\beta_1 = 0.85</math> for <math>f_c' \leq 30 \text{ N/mm}^2</math></p> <p style="text-align: right;"><math>a = \beta_1 c</math></p>
<p><b>AS3600</b></p>	 <p style="text-align: center;"><math>0.85 f_c'</math></p> <p style="text-align: right;"><math>\gamma = [0.85 - 0.007(f_c' - 28)]</math> <math>0.65 &lt; \gamma &lt; 0.85</math> <math>k_u \leq 0.4</math></p> <p style="text-align: right;"><math>\gamma k_u d</math></p>
<p><b>DIN1045</b></p>	 <p style="text-align: right;"><math>\beta_r</math></p> <p style="text-align: right;"><math>0.614x</math></p> <p style="text-align: left;"><math>\beta_s</math></p>

### 3.8 Design Example

This design example was selected from the reference Graded examples in reinforced concrete design (Dias, 1998), for the comparison of codes with respect to the design of a reinforced concrete beam section. This particular beam arrangement was chosen to avoid the code differences with respect to the analysis loading cases. Figure 3.6 shows the loading arrangements. Load values are at characteristic values and must have appropriate safety factors applied to them. The output from this exercise is given in Table 3.9. Wide disparities in output values can be observed.

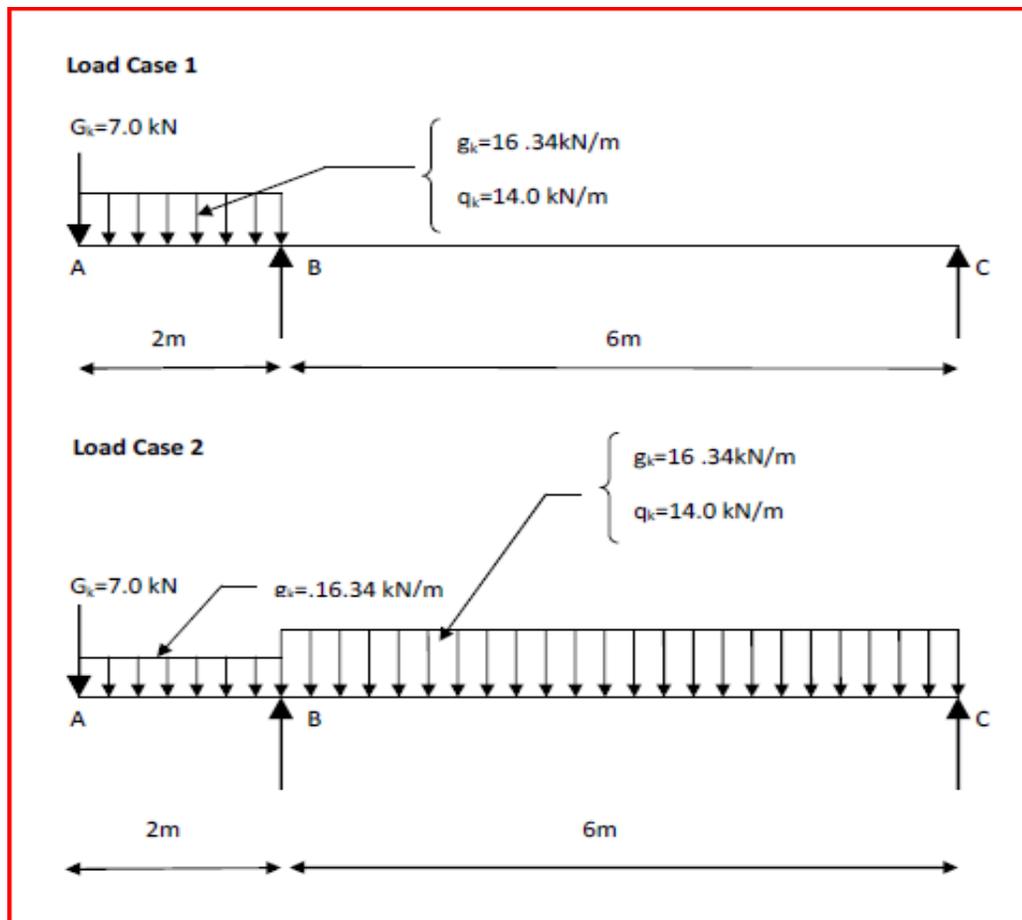


Fig. 3.6: Service loading arrangement (Example 7, of Graded Examples in reinforced concrete design by W.P.S.Dias,1998)

**Table 3.9: Summary of output of the design example**

Data-Item	BS8110	EC2	AS3600	ACI318	DIN1045
Depth	450mm	450mm	450mm	450mm	450mm
Span	6m	6m	6m	6m	6m
Cover	25mm	25mm	40mm	50mm	25mm
Hogging Moment	110 kNm	104.5 kNm	100 kNm	112.5 kNm	129.5 kNm
$A_s$	779mm <sup>2</sup>	768mm <sup>2</sup>	885mm <sup>2</sup>	796mm <sup>2</sup>	835mm <sup>2</sup>
Sagging Moment	181 kNm	171 kNm	148 kNm	187 kNm	200 kNm
Flange Width	1140mm	1140mm	1140mm	1140mm	1140mm
$A_s$	1224mm <sup>2</sup>	1176mm <sup>2</sup>	1235mm <sup>2</sup>	1255mm <sup>2</sup>	1225mm <sup>2</sup>
Max. Shear Force	154 kN	147 kN	130.4 kN	159 kN	180.25 kN
Design Shear Force	131 kN	124 kN	111.8 kN	135 kN	164 kN
Shear Capacity	0.55 N/mm <sup>2</sup>	0.38 N/mm <sup>2</sup>	0.45 N/mm <sup>2</sup>	0.63 N/mm <sup>2</sup>	-

When considering the above mentioned similarities and differences in different reinforced concrete standards, we observe that the data items of most of the standards are similar in nature but the methods of evaluating these data items differ. We could also observe that the building elements and their functions have been treated in by broadly similar fashion by different standards. Thus these similarities could be used in COIDS.

## CHAPTER 4

### COMMON INTERFACE CONCEPT AND IMPLEMENTATION

#### 4.1 Introduction

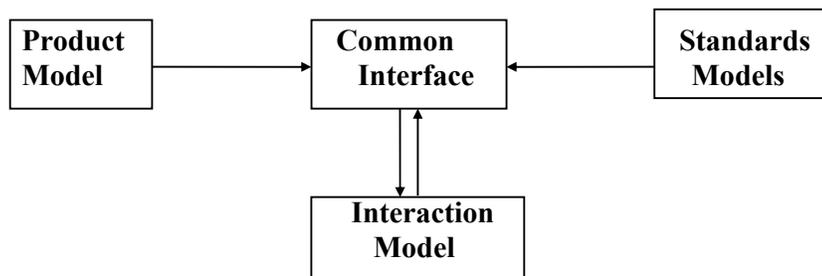
The Common Interface for Design Standards (COIDS) is called “common” because it is designed to accommodate several design standards. Since the Standards are upgraded with new knowledge, the program is designed to accommodate this flexibility. Hence, even a new standard could be plugged in to the common interface. The word “interface” is used because there are three modules (in fact called “models”) that interact with it (see Fig.4.1), namely the Standards Model, referred to above and which can contain more than one standard; the Product Model, which is a description of the structure; and the Interaction Model, through which interaction between models and between these and external entities (including user input and analysis software) is handled. The term “interface” is therefore not used in a narrow programming context, but rather as explained above.

One of the key aspects of the approach adopted here is that inputs, such as beam or column data input, are fed to the program only once – i.e. once the element data is entered for the analysis phase, it does not have to be entered again for the design checking phase. This is a feature of the Interaction Model.

The main feature of the Product Model is that its data is independent of the standards used to check the structure. The data consists of geometry, sectional properties and load data. Some load data, such as load combinations, will be standards specific and hence handled by the Interaction Model.

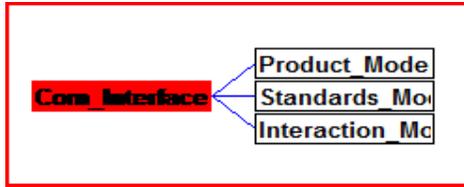
The Standards Model is the model that handles the standards that are included in the COIDS. One of the key features of the COIDS is that it represents the standards in the same format as in such standards. For example the Class Clauses represent Clauses of the standards, Class Symbols represents the Symbols of the standards and Class Table represents the Tables of the standards. The change of a clause or a table can be accommodated without major changes to the programme code.

The most significant feature of this approach is that in the Standards Model, the data processing will simulate the procedure followed by a practicing engineer – e.g. in checking a beam for flexure; the program will follow the code clauses similar to an engineer. In other words, the program procedures are not “hard coded” and code clauses are encoded as separate methods. A user request for a particular design check will be processed by picking the right clauses and tables to execute the request. This is the main innovation in this work. This approach has great practical efficiency, in that code changes can be very easily accommodated. In addition, the notion of staying as close as possible to the way human experts work can be seen as being conceptually “authentic”.



**Fig. 4.1: Conceptual Model of the COIDS**

The COIDS concept is implemented in the object oriented shell, KAPPA, described in detail in Chapter 5. However, screen images from KAPPA are used in this chapter too, as they are often useful for describing the structure of the common interface COIDS. For example, Fig. 4.2 shows how the three different models are linked to the common interface object.



**Fig. 4.2: KAPPA representation of the Conceptual Model of the COIDS as an Object Hierarchy**

#### **4.2 Product Model**

The Product Model represents product data, i.e. the frame data, such as node data, element data, support properties, and section properties. The basic product data is fed to COIDS through the basic data input session window, details of which are given in Chapter 6. Based on this data input, the corresponding KAPPA Product Model object hierarchy instances are dynamically generated (see Fig. 4.3 and Fig. 4.4). It holds the classes such as Nodes, Elements, Supports and Section properties. Load data is included within the Element class. This model is independent of design standards. The methods to generate the necessary instances are included in both classes Nodes and Elements.

The frame data is fed only in one session, unlike the current practice where the element data needs to be entered in the analysis session and during the design session. The data items like element dimensional properties, material properties and element loadings need to be entered during this session. This input data is stored in the slots of the Product Data Instances. The data will be used by the Interaction Model of COIDS to generate an input file for the Analysis package (MFEAP).

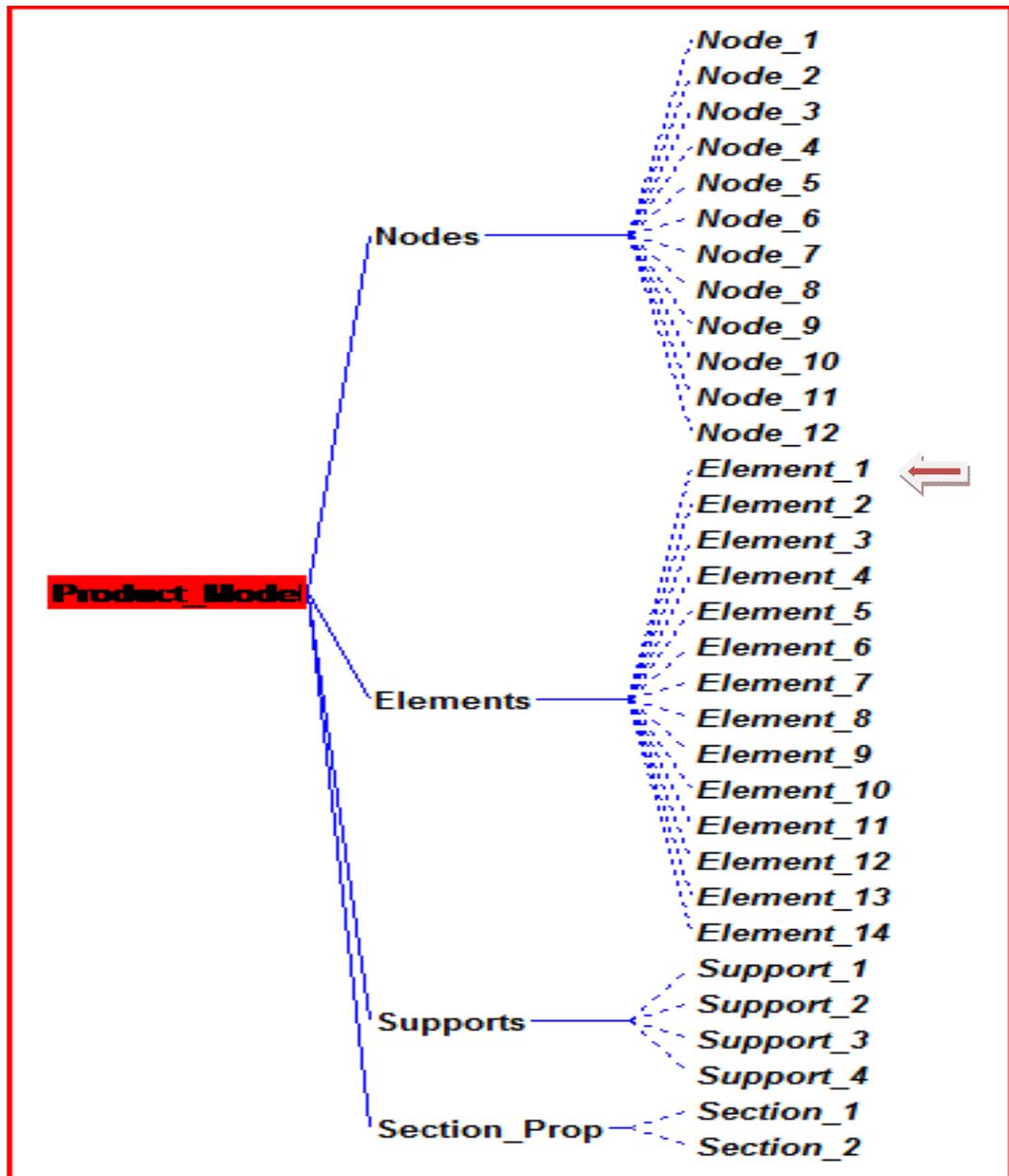
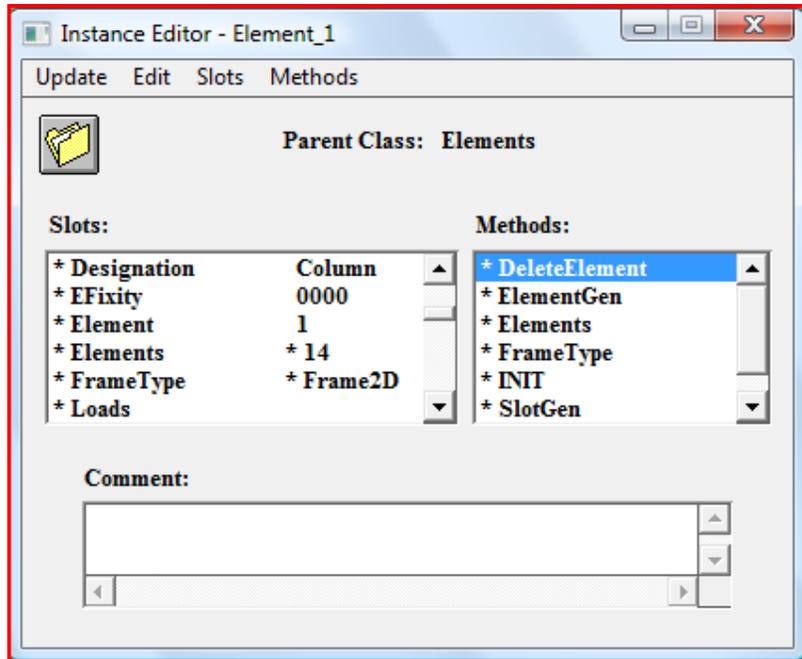


Fig. 4.3: Product Data Object Hierarchy



**Fig. 4.4: Typical Element Instance**

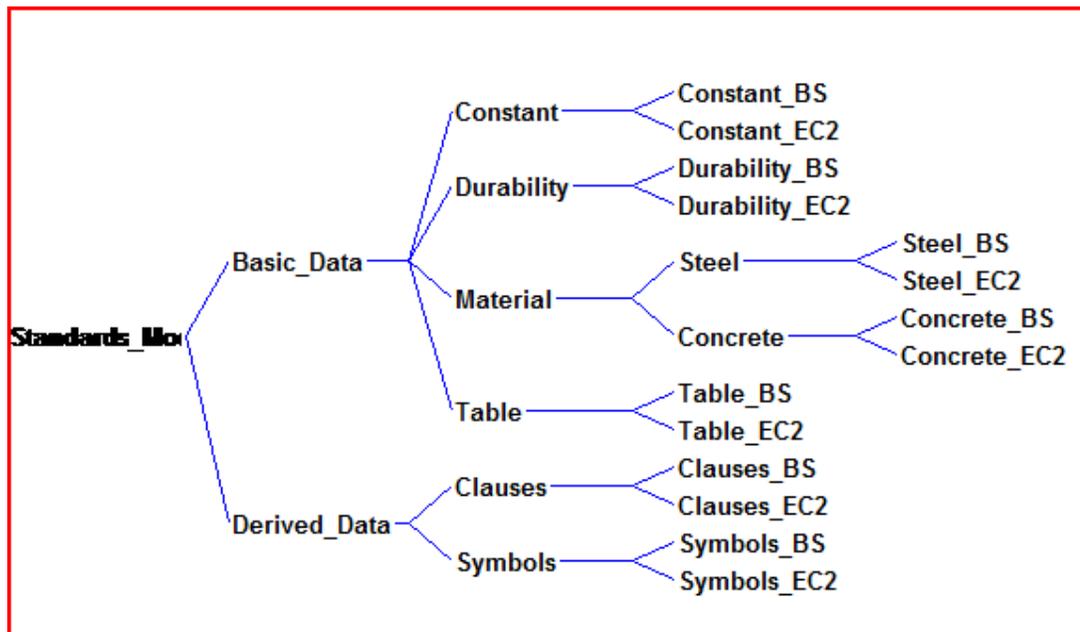
### 4.3. Standards Model

The Standards Model represents the data items in the standards. This class is designed to hold several standards. Thus the development of this model has considered common features of different standards and their basic structure so that many codes could be accommodated in COIDS. Chapter 3 identifies the common features of codes and their basic structure. According to Chapter 3, there are many differences in different standards, when considering the features at the micro-scale, but at the macro-scale, they all follow the same design philosophy - that is the limit state philosophy and the same design intents such as durability considerations and fire considerations. This common feature is captured in designing this model.

There are two main sub classes (see Fig. 4.5) in this model, namely class Basic Data and class Derived Data. The class Basic Data object represents data item types that do not need any equation or procedure to derive them, and include constants, durability data, material data and tables.

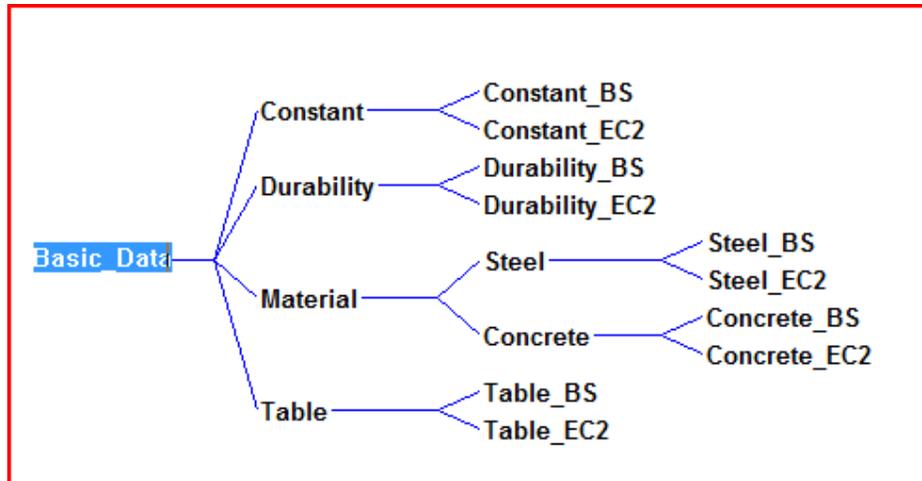
The Derived data object represents data items that need equations or a procedure to derive them. Each of them will have a different form of evaluation - e.g. by executing sets of equations, or through inferencing using sets of rules. The main Objects in this hierarchy are Clauses and Symbols classes. This hierarchy is standard dependent. A new standard can be plugged in as a subclass to the appropriate class objects.

Two features of the Standards Model hierarchy are that (i) it does not have any instances, as in the Product Data Hierarchy; and that (ii) many of the classes are empty, apart from the “leaf” classes – i.e. those at the end (or bottom) of the hierarchy, which are code dependent. The hierarchy then serves mostly as a classifier. It is also a structure that can incorporate any common features if any, as demonstrated below.



**Fig. 4.5: Standards Data Object Hierarchy**

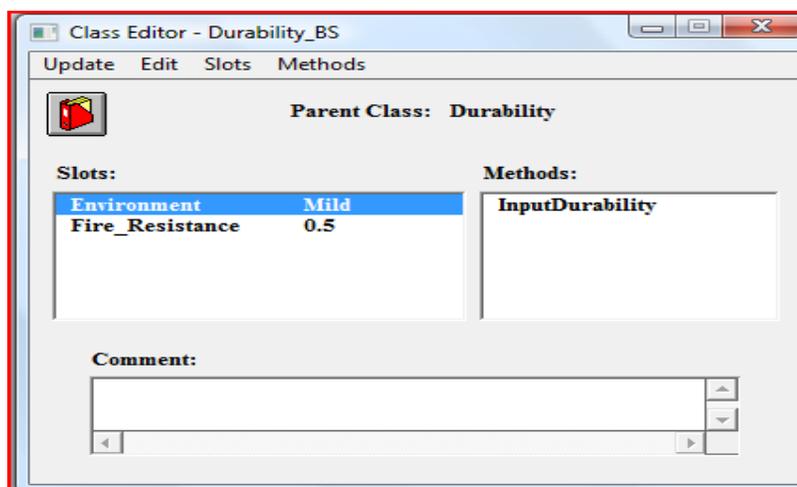
**4.3.1. Basic Data hierarchy:** Class Basic Data is a root class for all the data items that do not have an explicit method of evaluation expressed within the standards. Values for these data items are either provided in the standard or by the user. It holds the subclasses such as Constant, Durability, Material and Tables (see Fig. 4.6).



**Fig. 4.6: Basic Data Object Hierarchy**

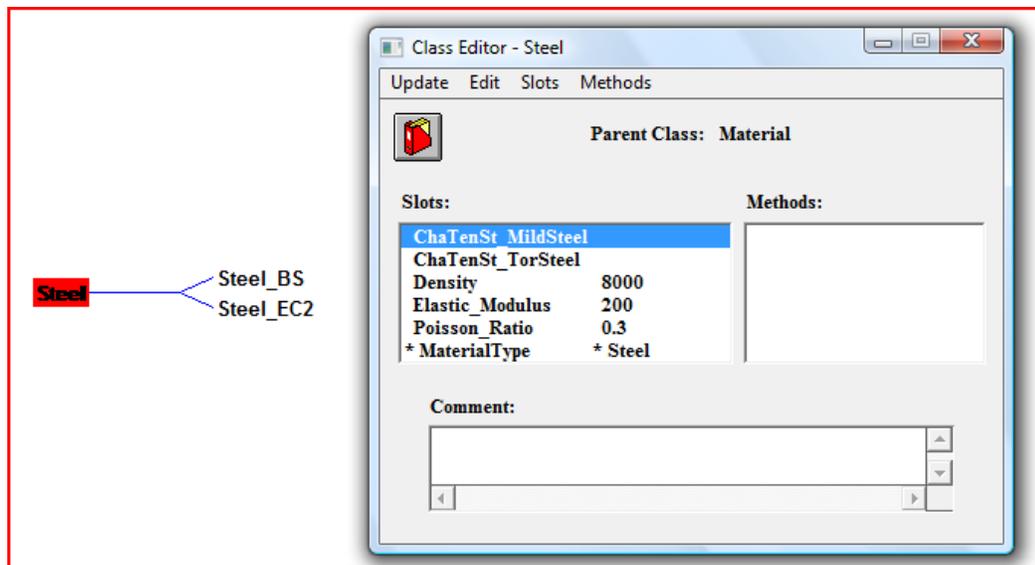
**Class Constant:** A constant can be of any data type such as numeric, symbolic, Boolean or linguistic. An example is a constant such as  $\alpha$ , the angle assumed for the compression strut in the shear resistance calculation for bent up bars in BS 8110.

**Class Durability:** The durability data refers to requirements, such as the exposure class “mild” in BS 8110. The class also contains the fire resistance (another type of durability) requirements in required exposure duration in hours (see Fig. 4.7). These are provided by the user at the design checking stage. The user may be assisted by a dialog box which will be displayed at run time.

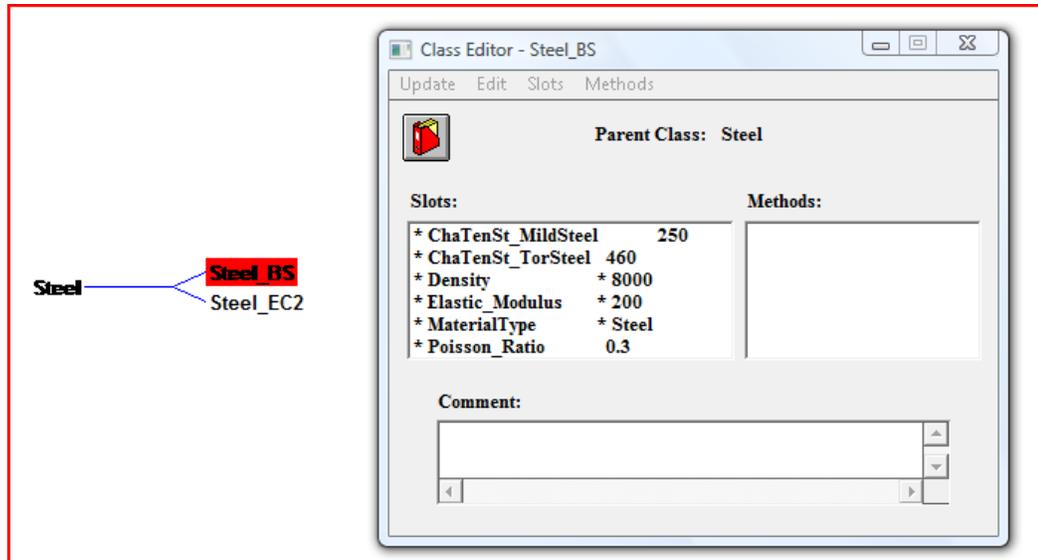


**Fig. 4.7: Sub Class Durability\_BS**

**Class Material:** The materials data are provided by the user. The class Material Data has two subclasses for Steel and Concrete. These contain material properties such as density and E value. It should be noted that strength will vary from code to code. For example, BS 8110 specifies steel strength as 460 MPa and EC2 as 500 MPa. Also, concrete strength is specified as cube strength in BS 8110 and cylinder strength in EC2. Hence, they will have to be stored in the code specific subclasses such as Steel\_BS, Steel\_EC2, Concrete\_BS and Concrete\_EC2 see Fig.4.8 and Fig. 4.9 for the difference between the Steel and the Steel\_BS classes. The latter inherits the values of Elastic modulus and density from the former, but the value of strength is declared only in the latter.



**Fig. 4.8: Common Steel Class**



**Fig. 4.9: Steel\_BS Sub Class**

**Class Table:** The class Table represents those data items whose values are obtained from lookup tables within the standard. This will hold the methods like table lookup routines and methods for interpolation etc. As there is no facility to represent a table in KAPPA we represent these tables in a spreadsheet (Lotus123 or Excel) or in a database. The access to these data files is through the DDE (Dynamic Data Exchange) facility provided by KAPPA. Fig. 4.10 shows how Table 3.4 of BS 8110 represented in a spreadsheet, and Fig. 4.11 the way it is implemented in KAPPA. It should be noted that the same class Table\_BS can be used to process information from more than one table – in this case Table 3.4 for long term durability and Table 3.5 for fire resistance – by using a different method for each. The slots that are included in the class Table is to temporarily store the rows and column values that are needed to locate the value in the table. Note also note that Table can contain both numeric and alpha-numeric values.

BS 8110 - Part 1 : 1985

Table 3.4 Nominal cover to all reinforcement ( including links ) to meet durability requirements .

Condition of exposure

	mm	mm	mm	mm	mm
Mild	25	20	20	20	20
Moderate	NULL	35	30	25	20
Severe	NULL	NULL	40	30	25
Very Severe	NULL	NULL	50	40	30
Extrem	NULL	NULL	NULL	60	50
Max. w/c	0.65	0.6	0.55	0.5	0.45
Min. Cement	275	300	325	350	400
Con. Grade	30	35	40	45	50

Fig. 4.10: Lotus 123 representation of the BS Table 3.4 (BS34)

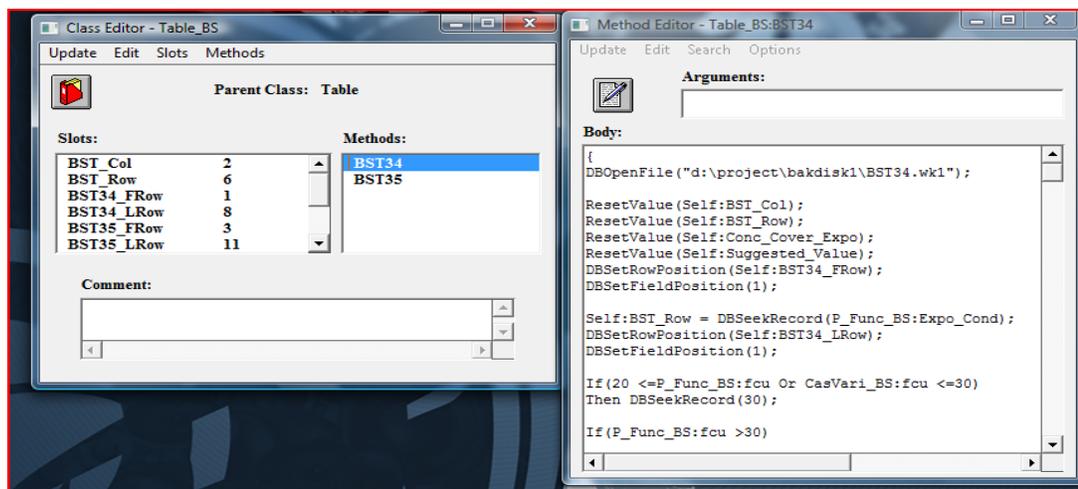
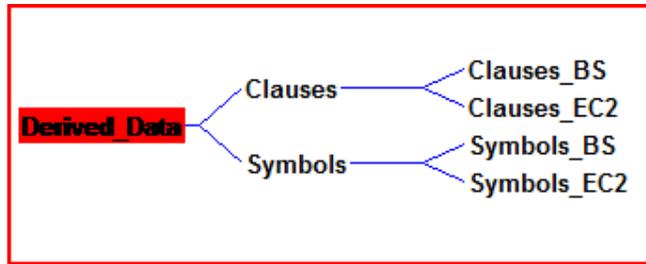


Fig. 4.11: Sub Class Table\_BS and the Method BS34 look up routines

**4.3.2 Derived Data hierarchy:** Class Derived Data is a root class for all the data items that have explicit methods of evaluation expressed within the standards. This root class has two main subclasses, namely Clauses and Symbols (see Fig, 4.12). The main standards features are included in this hierarchy, namely clauses and symbols. As explained above, COIDS attempts to encode the design standards in the same high level format as in the standards. It is even possible to include the text format into the method, and the “post message” command will inform the user during the checking mode if any clause is violated. However this hybrid approach is only partially implemented in COIDS to demonstrate the concept.



**Fig. 4.12: Derived Data Object Hierarchy**

**Class Clauses:** This is the parent class of Clauses BS and Clauses EC2, which are the code dependent classes. All clauses, as explained in Chapter 3, consist of an “If” part and a “Then” part. The “If” part will specify the condition to execute the “Then” part of the clause. Thus a programme can be written for each clause to execute the instructions of the clauses. Following standards features were utilized to structure the class Clause.

- It was observed that there is a common procedure adopted for designing and checking elements by different standards. For example in order to calculate the design resistance moment of a flanged beam, first the effective span of the beam needs to be calculated, then the flange width of the beam needs to be calculate and thereafter equations are provided to calculate the design resistance moment.
- There are one or more clauses needed to find a particular data item. When the data item needs more than one clause to evaluate the value, a set of clauses needs to be subjected to inferencing; for example to calculate the effective span of a beam, according to clause BS 8110, Clause numbers 3.4.1.2 -3.4.1.4 are needed.

Since most standards follow a similar procedure, class Clauses holds common slots to hold the data items needed for inferencing (see Fig. 4.13).

Clauses BS will include all the British Standard’s clauses as methods (see Fig. 4.14), which are executed on the instructions (messages) send by the Interaction Model based on the user request. The slots of Clauses\_BS holds the clause numbers that refer to KAPPA methods, which include the program code for the method (see fig. 4.15). The methods are not hard coded, and are independent of each other,

representing clauses by their clause numbers. For example clause 3.4.1.2 of BS 8110, is represented as a method in the object Clauses\_BS as BS3412. This representation of code clauses makes it much easier to change the programme following standards revisions.

The slots seen in the Clauses\_BS are the data items that need to be evaluated by inferencing. They hold the clause numbers that will be subjected to inferencing (see Fig. 4.16).

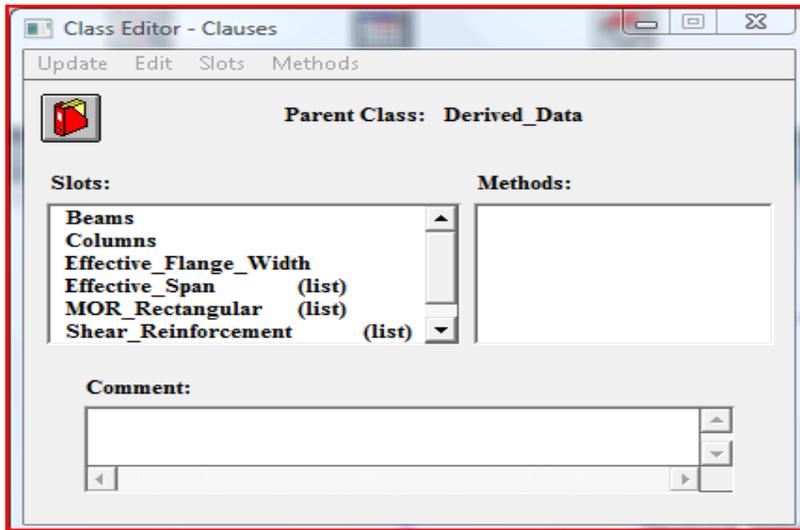


Fig. 4.13: Class Clause Object which includes the common slots clauses

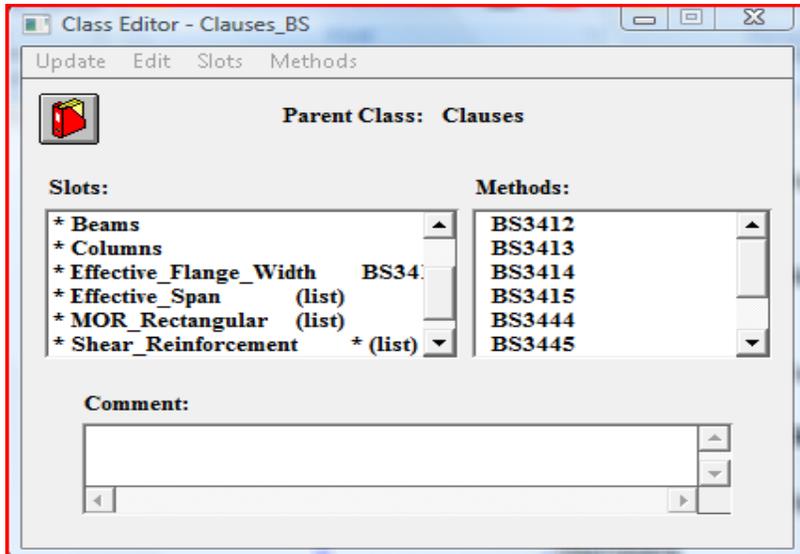


Fig. 4.14: Sub Class Clause\_BS Object: slots for inferencing and BS 8110 clauses as methods

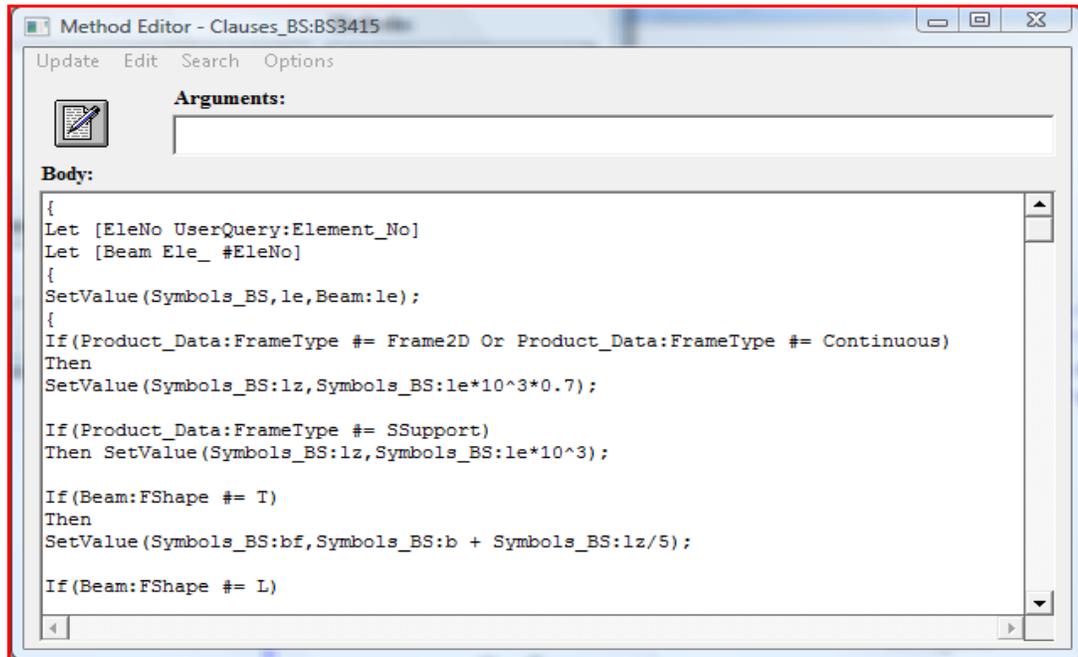


Fig. 4.15: KAPPA Method BS3415 represents the BS8110 clause 3.4.1.5

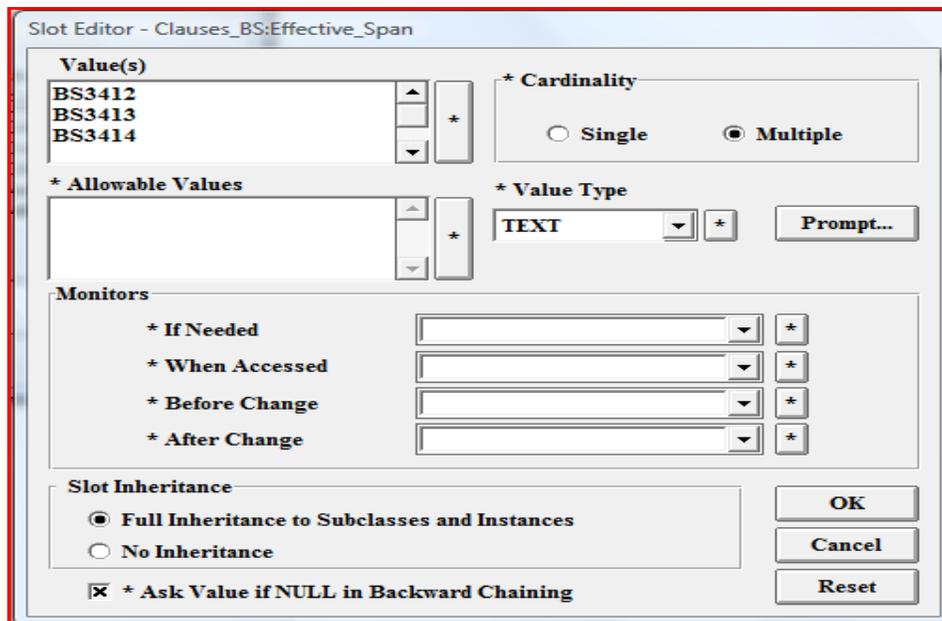
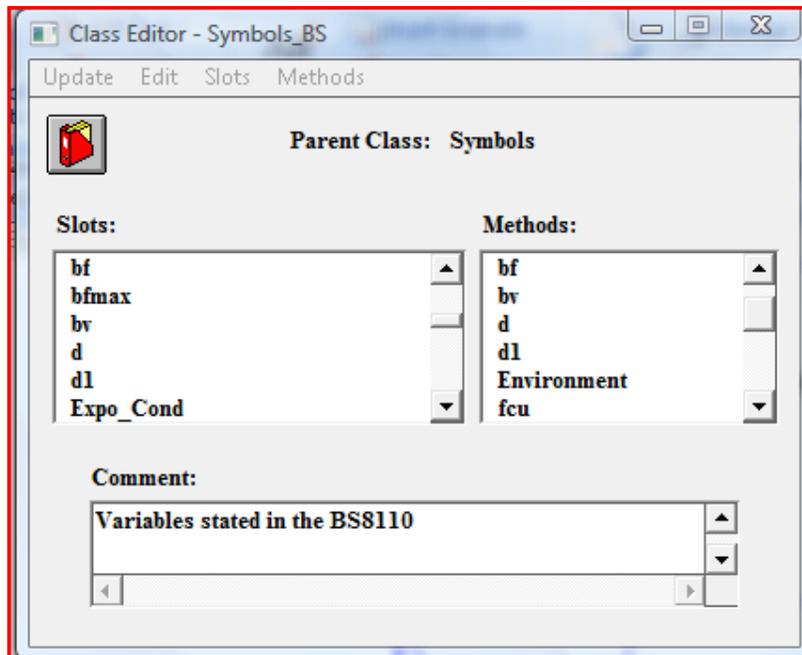


Fig. 4.16: Clause\_BS effective\_span slots

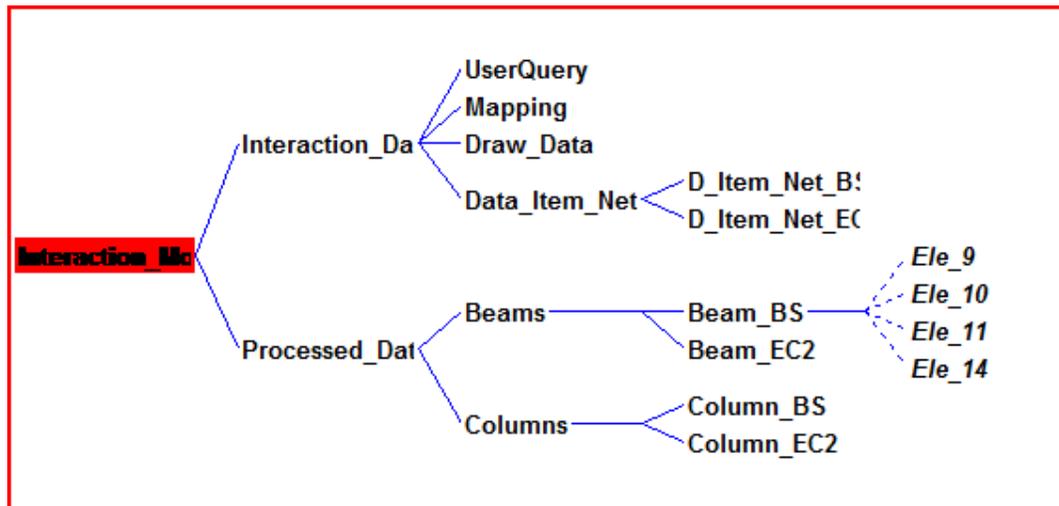
**Class Symbols:** This is the parent class of the Class Symbols BS and Class Symbols EC2. This class holds only the common initialization method to initialize the KAPPA slot values, since there are no common symbols among different standards. The data items that are defined under symbols are handled by these classes. Most of these data items do not have specific clauses to derive their values. They may be a user input or an item that can be executed using a simple equation (no clauses). For example “As” is defined in BS8110 as area of tension reinforcement, which is a user input and “d” is defined in BS8110 as effective depth of the tension reinforcement, which can be calculated by deducting from the overall depth the cover, diameter of the shear link and half of the diameter of the tension reinforcement. Some of the symbols such as “v”, defined in BS8110 as design shear stress at a rectangular cross section, have calculation procedures (in this case in Cl 3.4.5.2 of BS8110). The Class Symbols\_BS slots hold the values of the symbols temporarily during the checking mode and the KAPPA methods in the same class are used to derive the corresponding value. The processing of the values will be dynamically derived based on the user request (see Fig. 4.17).



**Fig. 4.17: Sub Class Symbol BS, which handles the Symbols data Items in the Object’s Slots and the corresponding methods in the Object’s Methods**

## 4.4 Interaction Model

The Interaction Model is the main module that links the user and the common interface “COIDS”. User Interface session windows and the other modules such as the Product Model and the Standards Model are linked to this module. All the messages are generated from the Interaction Model Objects. Interaction Model consists of two main sub classes namely; Interaction Data and Processed Data objects (see Fig. 4.18).



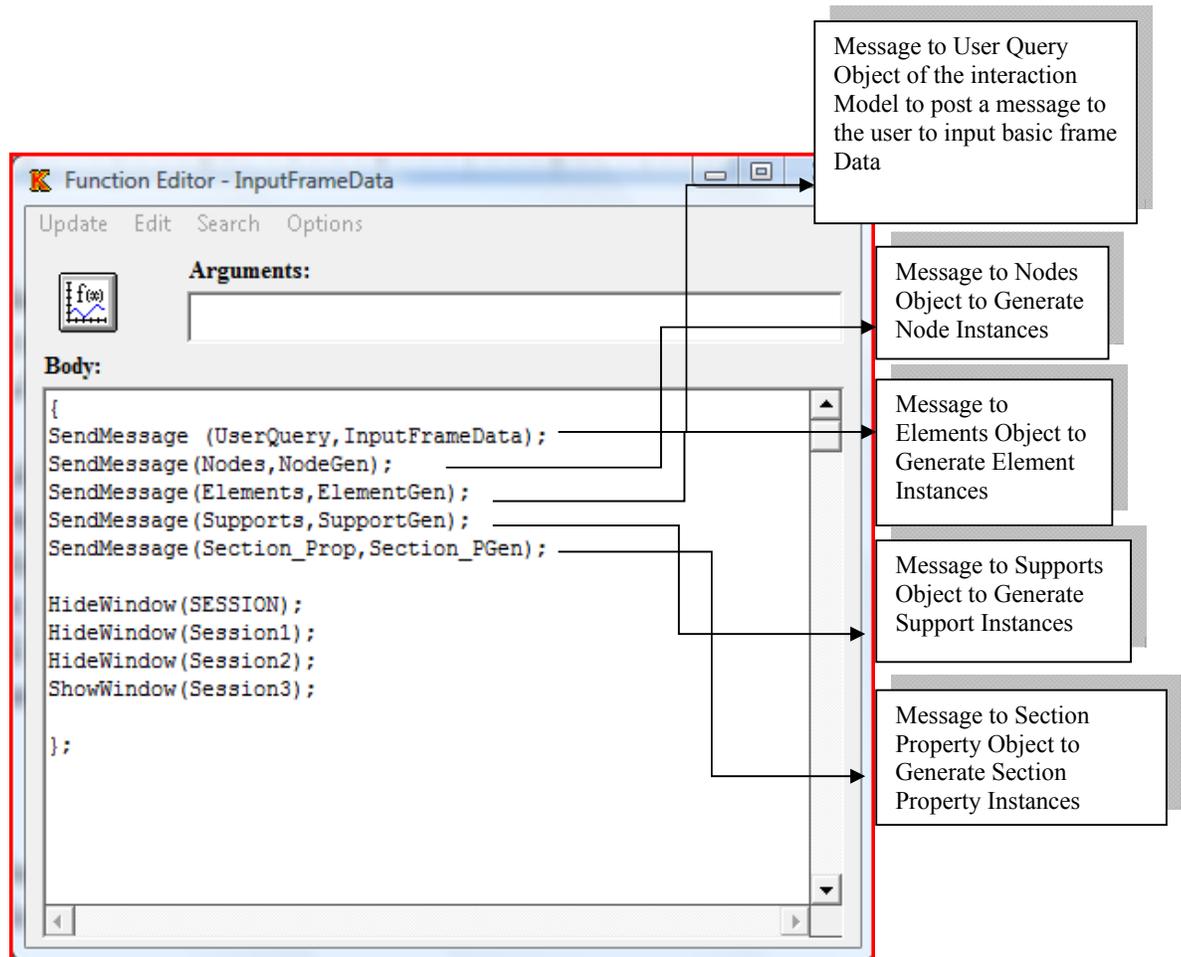
**Fig. 4.18: Interaction Model Object hierarchy**

**4.4.1 Class Interaction Data hierarchy:** This object hierarchy, as the name implies, will interact with the user and the other COIDS objects. The object hierarchy includes objects that communicate with the user and external programs, access and process other data files and communicate with the internal objects of COIDS.

**Class UserQuery:** The main task of this object will be to communicate with the user through KAPPA dialog boxes. Initially the user needs to give the product model data such as Frame data, Node data, Element data, Support data and Section Properties. The main KAPPA session windows discussed earlier in this chapter, will lead the user by the buttons included in the window. These buttons are linked to the KAPPA functions, which include methods to send the messages to the COIDS objects. For example when the user presses the frame data input button, messages will be originated from the KAPPA function editor (further discussed in Chapter 5) to the

COIDS UserQuery object to post a dialog box to obtain the initial frame parameters such as job name, frame type, number of nodes, number of elements, number of supports and number of section properties.

Based on the user input, KAPPA Function “Input Frame Data” will send messages to corresponding objects to generate instances (see Fig. 4.19). The User Query object also posts dialog boxes to the user for input data to the Product Model Instances (see Fig. 4.20), and a typical durability data input dialog box is shown in figure 4.21. The slots of the UserQuery object are to hold the main user inputs such as frame data inputs, analysis software execution file location list and the user’s element check condition lists.



**Fig. 4.19: KAPPA Function’s Input Frame Data Method**

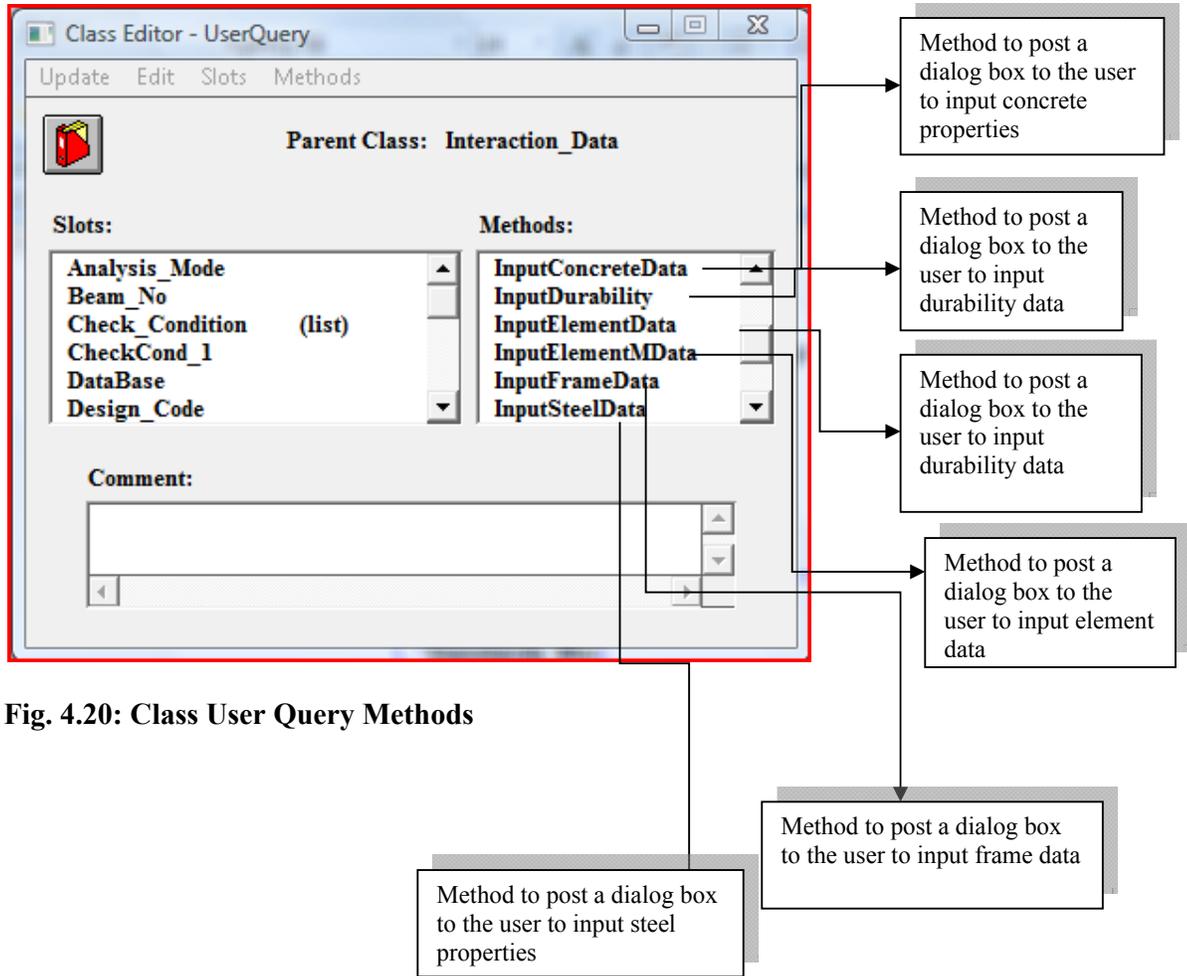


Fig. 4.20: Class User Query Methods

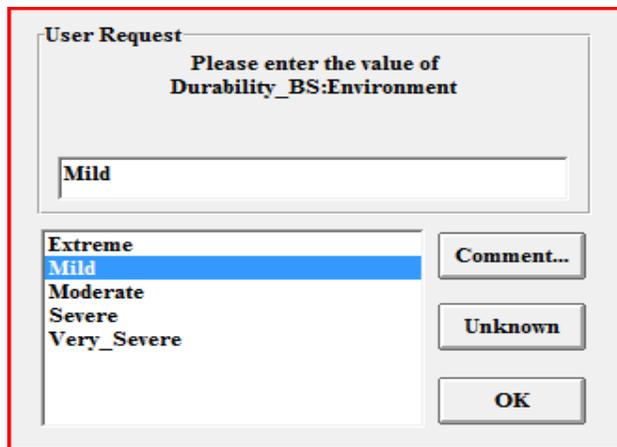
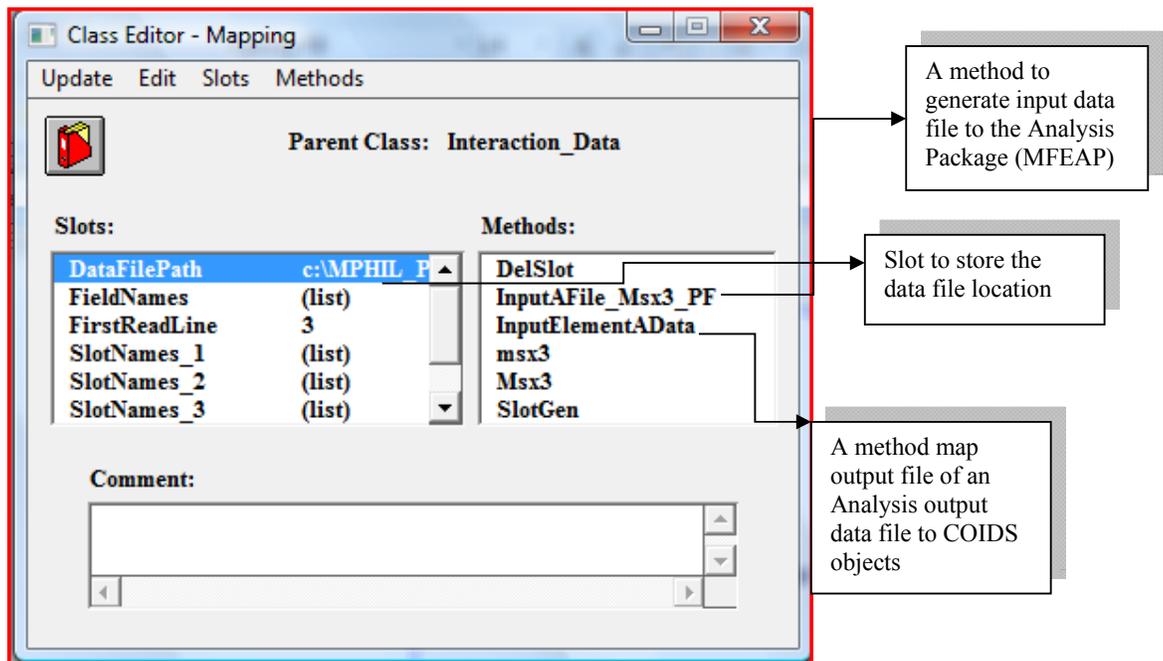


Fig. 4.21: Durability Data dialog box posted to the user by the User Query object

**Class Mapping:** The task of this object is create the input file for the Analysis Package (MFEAP) using COIDS object data which is fed by the user, and to map the MFEAP output data file (see Fig. 4.22) to instances created under the class Processed\_ Data, such as element axial forces, bending moments and shear forces. The slots in this object will hold the data file locations and the field names, row and column numbers to generate the input file for the analysis package and to map the output data file to the COIDS instances.



**Fig. 4.22: Class Mapping**

**Class Draw\_Data:** This object handles the graphical user interface (GUI) of COIDS. In today's programming context the graphical user interfaces are an important aspect, since the graphical user interface will give an opportunity for the user to check the input and graphically observe the output result. We may note that the output is a direct result of the input. COIDS also has a capability to display the input of the user (see Fig. 4.23). This facility could be further developed to display the output results too.

Class Draw\_Data is independent of standards, and holds the methods such as show nodes and show elements to display nodes and elements respectively (see Fig. 4.24). The slots of this object temporarily hold the node numbers and their co-ordinates, and

element numbers, which will be utilized by the above show nodes and show elements methods.

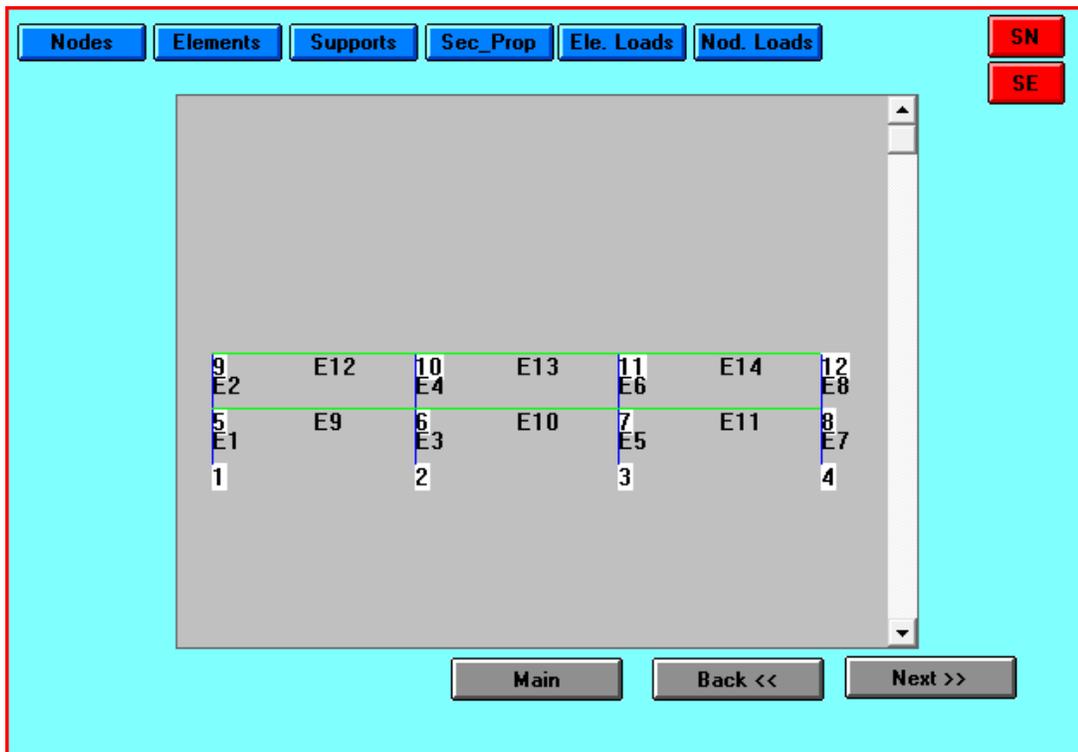


Fig. 4.23: COIDS Graphical User Interface

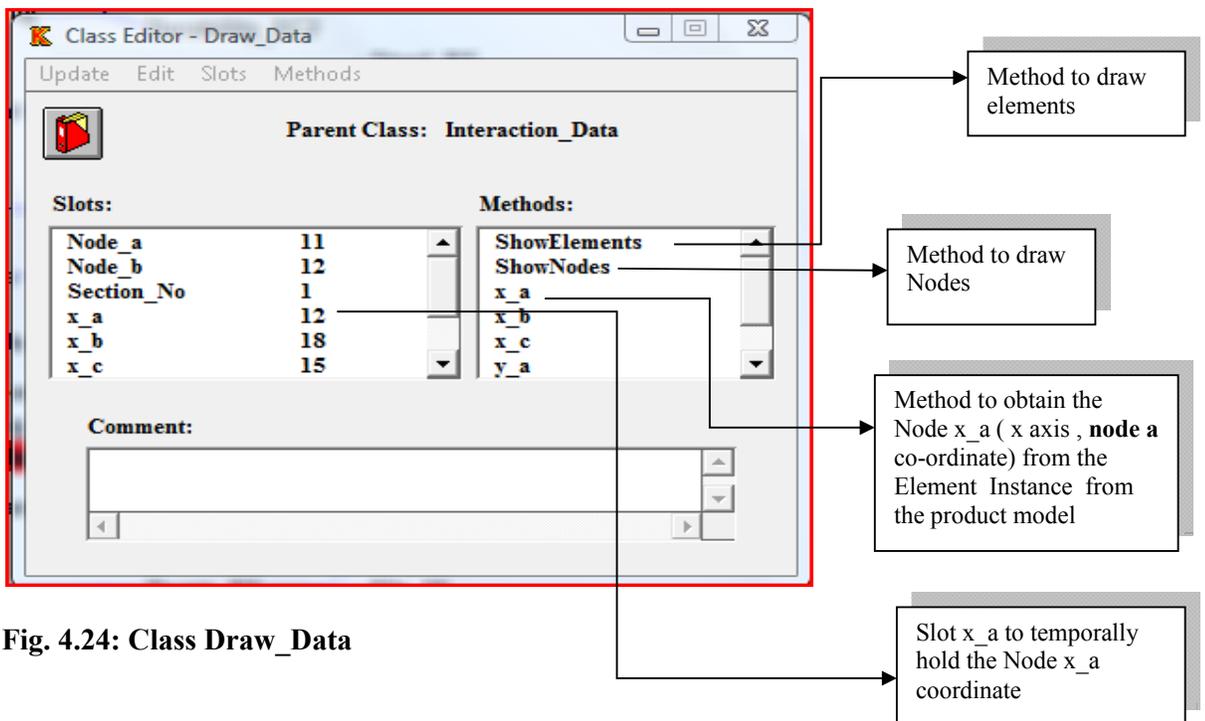
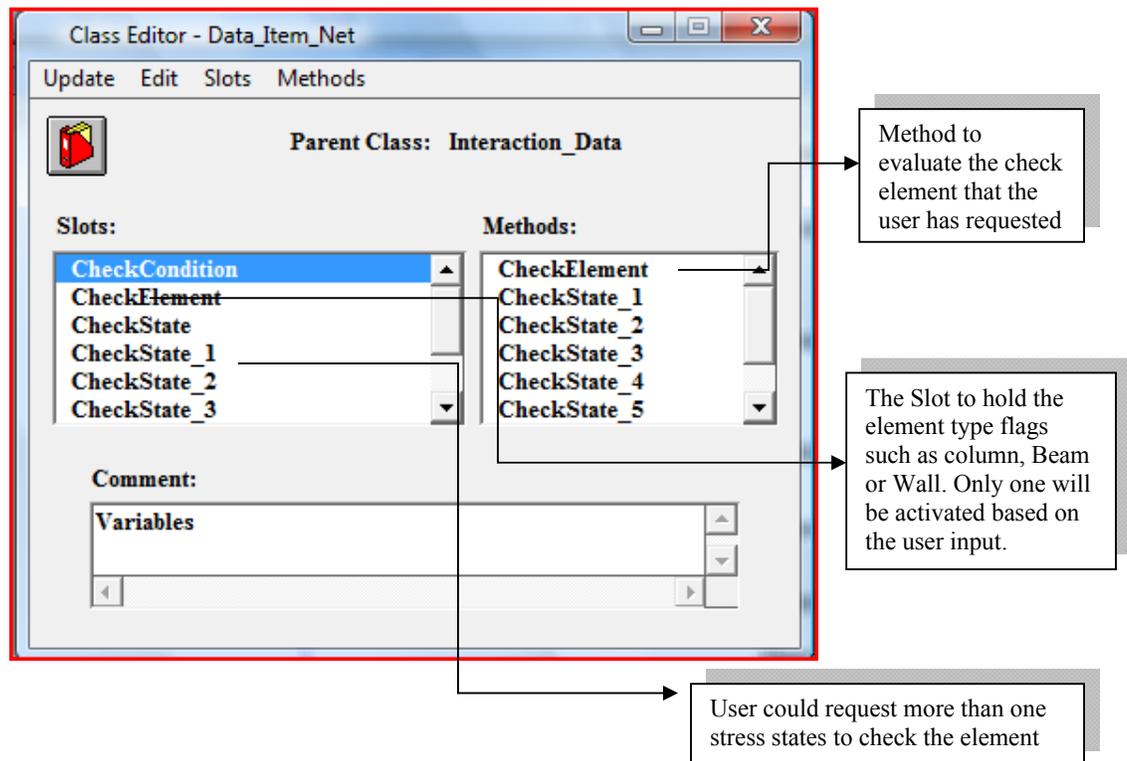
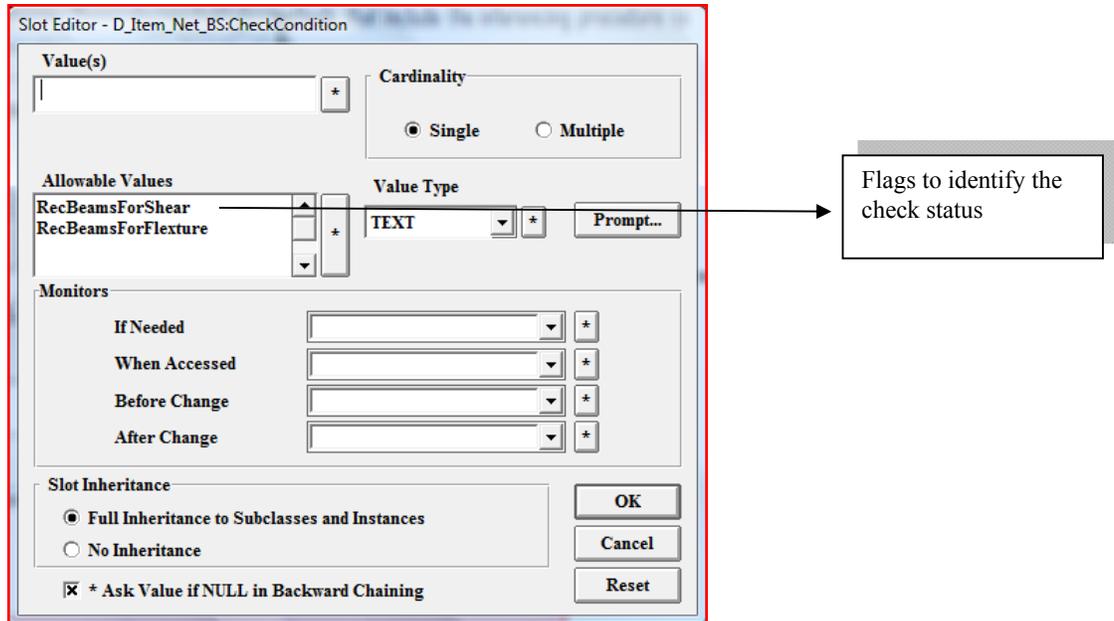


Fig. 4.24: Class Draw\_Data

**Class Data\_Item\_Network:** This is the main class that handles the checking mode of COIDS. The capacity checking of elements is dependent on standards; thus there are standards dependent sub-classes. These classes hold the methods that include the inferencing procedure to evaluate the user request (see Fig. 4.25). In checking an element complying to a particular standard for a user requested stress state, the programme needs to identify the member (whether it is a beam or a column), the standard that needs to be complied with, and the stress state that the element is to be checked for. Based on the user request, COIDS generates flags to identify the request. For example, if a rectangular beam is to be checked for flexure according to British Standards, a “RecBeamForFlexure” flag will be used (see Fig. 4.26). User has the option to define more than one check state, which will be stored in the corresponding slots of the class Data Item Network. Since most standards have a common approach for evaluating the frame elements, most of the methods and the slots are inherited by the sub classes Data Item Net\_BS and Data Item Net\_EC2.



**Fig. 4.25: Class Data Item Network**

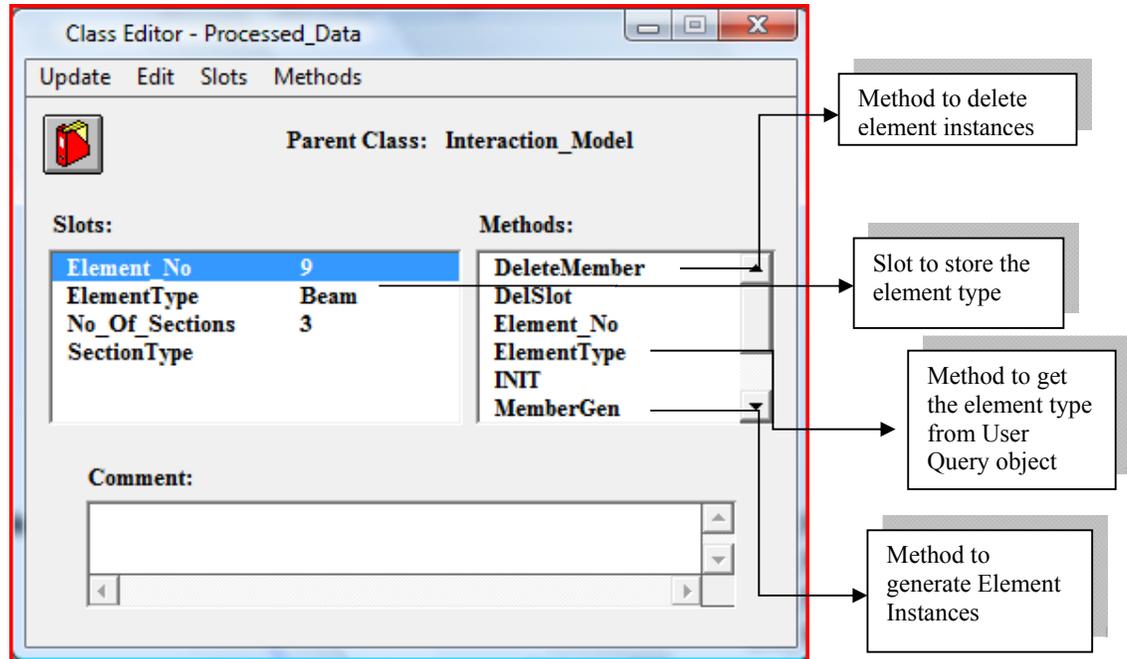


**Fig. 4.26: Class Data Item Network, slot Check Condition stores flags to identify the check state**

**4.4.2 Class Processed\_Data:** As the name implies Class Processed\_Data will handle the processed data of the COIDS. Element classes such as Beams and Columns serve as sub classes. These classes will hold the processed data of the respective element instances, which will be generated based on the user request. Class Processed\_Data will have the common methods to generate element instances and initialize slots and instances. The slots of the class Processed\_Data will hold the data items such as element number and element type, which are used to generate the element instances under the Beam\_BS or Beam\_EC objects (see Fig. 4.27). The sub classes Beams and Columns handle element dependent (but standards independent) data items. Standards dependent data items are introduced at the next level, e.g. Beam\_BS or Column\_BS. Class Beams will handle common data items such as width “b” and the height “h” of the beams ( see fig. 4.28), while sub class Beam\_BS will handle the code specific data items such as effective span “le” (see Fig. 4.29). The effective length of the beam will be evaluated by inferencing the clauses (written as methods) that are stored in a list in the slot “effective span” in the class Clauses\_BS. The class Beam\_BS will also hold the steel provided by the user and the calculated minimum steel required by the British Standards. The comparison of the provided and the minimum requirements will indicate whether the element conforms to the standards or violates any code clauses.

It should be noted that the element instances in the Processed Data part of COIDS are dynamic in nature and “exist” only when that element is being checked. Data is imported to it from various other parts of COIDS. It also interacts with the clauses in the Standards Model. Some typical interactions are described below:

1. Relevant data is imported from the (static) instance of the same element in the Product Model such as width and height, already defined when the Product Model is generated.
2. Data such as Bending Moments and Shear Forces from the output data file of the Analysis Package (held outside the KAPPA environment) are obtained through the Mapping object.
3. Data for checking such as reinforcement provided can be obtained from the User Query object.
4. The actual checking itself is carried out by accessing the clauses written as methods in the Standards Model.



**Fig. 4.27: Class Processed Data**

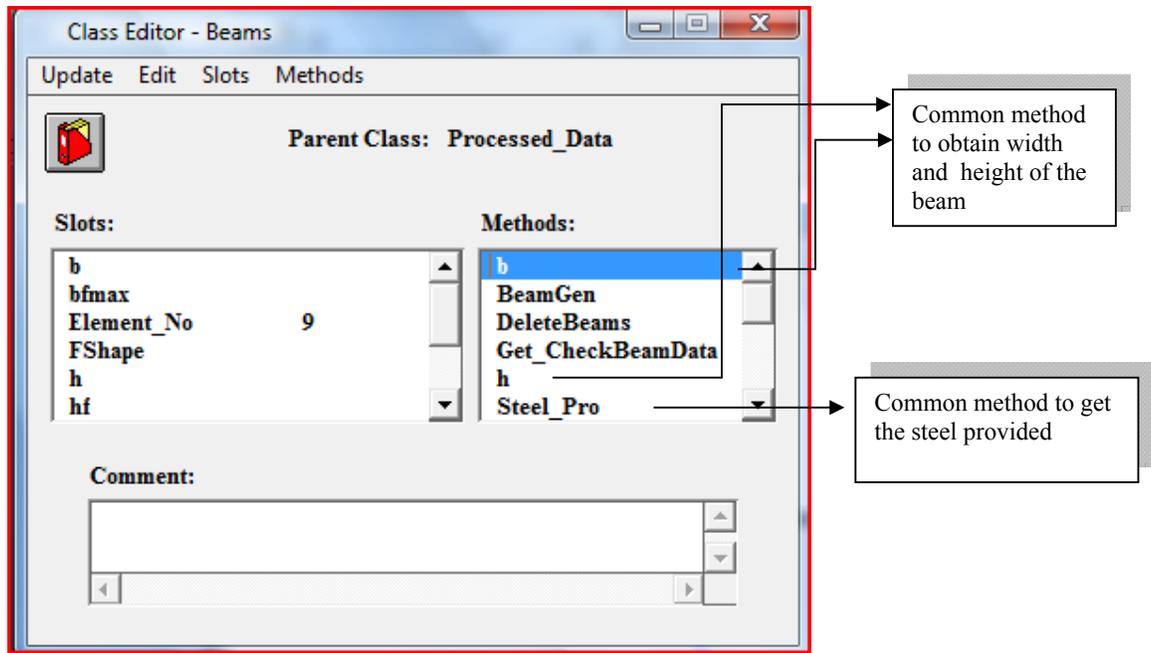


Fig. 4.28: Class Beams

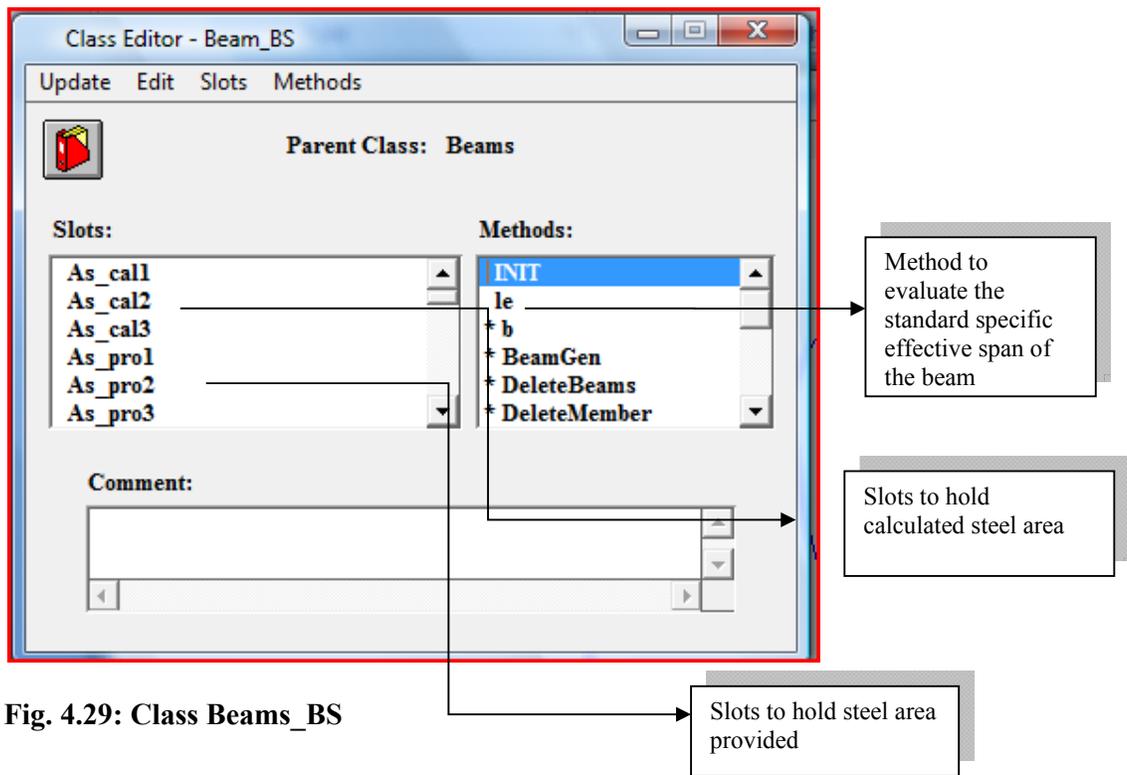


Fig. 4.29: Class Beams\_BS

## CHAPTER 5

### KAPPA APPLICATION DEVELOPMENT SOFTWARE

#### 5.1 Introduction

This chapter highlights the features of KAPPA and how the KAPPA facilities are used to develop knowledge based applications such as COIDS. The main source of reference to this chapter is from KAPPA Manuals.

KAPPA is used to build knowledge-based applications, systems that capture the knowledge necessary to understand some complex system or domain. Building a knowledge-based system means building a realistic model of the actual system. Knowledge-based systems are used for tasks such as planning, diagnosis, design, scheduling, training, data interpretation (processing) and configuration.

KAPPA knowledge bases are built around the important components, behaviour and relationships in a system. A good candidate for a KAPPA knowledge-based system is one where you understand how the components interact, but where there is too much complexity for one to predict the behaviour of the entire system. Another good candidate for a KAPPA knowledge base is a situation where there is a the need to distribute the knowledge of a few experts to a broader group of people.

**5.1.1 Introducing KAPPA Knowledge Elements:** KAPPA provides a wide range of tools for constructing and using knowledge-based systems. Since KAPPA is an Object oriented programming shell, the components of the domain are represented by the object oriented structures called objects, which can be a class or an instance. They can represent physical things like “Beams” and “Column” or concepts such as “Durability”.

The relationships among the objects in a model can be represented by linking them together into a structure called a “hierarchy”. Object oriented programming tools within KAPPA will help the user to create objects and the object hierarchy, write “methods” to specify the behaviour of the objects, and create “slots” to represent the properties of objects.

Once the real world problem is represented in the object oriented format, then a set of rules can be specified to get the desired outputs from the complex system that was specified earlier.

In rule-based programming, each rule specifies a set of conditions and a set of conclusions to be made if the conditions are true. Each rule is a relatively independent module, thus a reasoning systems could be built gradually, rule by rule.

Rules can be set in two basic ways, “forward chaining” and “backward chaining”. In forward chaining, conclusions are drawn from known facts and these conclusions become facts from which to draw further conclusions, for example if there are several clauses that need to be used to evaluate a data item (i.e. there are several rules to determine the data items), the those clauses (rules) need to be forward chained. In backward chaining, a desired conclusion is specified and the conditions of the rules are used to determine if the conclusion is true or how it may be made true. Backward chaining is used as the “Goal” finding mechanism or the programme ending mechanism.

**5.1.2 KAPPA Applications:** Applications written in KAPPA can perform two important tasks;

1. Help human decision making: Thus, the application should enable the user to understand how a result or a proposed decision was derived.
2. Using as a learning tool: A user can examine parts of a domain that are inaccessible to the user in the real world, and conduct experiments which are dangerous, expensive, impractical, or impossible in reality.

In order to observe and control the operation of a knowledge base, we can also use a variety of graphic images in building the KAPPA interface. The Active Images package contains buttons, bitmaps, drawings, state boxes, meters, line plots and sliders (see figs. 5.5- 5.6). These indicators can be used to display the values of important parameters and observe how they change while the system is in operation. These Active Images assist the user to control the object oriented model. These images are linked to the KAPPA Functions, which will be discussed in detail later in this chapter.

The KAPPA programming language, KAL is used to develop the knowledge base. KAL language is used to develop KAPPA's rules, methods and functions. Adding and retrieving information to the knowledge base is also done using KAPPA. It is called a 4<sup>th</sup> generation language since its syntax is fairly close to the English Language. For example, to send a message to a particular object, the KAL syntax is “ SendMessage (Object Name, Method Name)”. The root language used to develop KAL is C language, thus the KAPPA application development system uses C source code. This allows KAPPA to extend its capabilities.

## 5.2 The KAPPA Interface

KAPPA provides a powerful interface for the application developer. The interfaces consist of tools for viewing and modifying the various KAPPA elements. The KAPPA interface also contains tools to build, customized displays and browsers. The KAPPA Main Window consists of several icons (see Fig. 5.1). Each icon represents one of the windows in KAPPA:

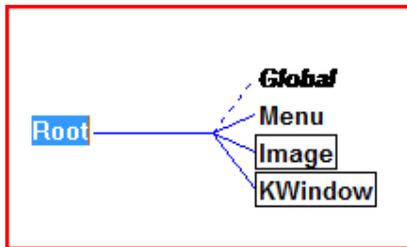
- Object Browser
- Session Window
- Editing Tools
- KAL Interpreter
- KAL View Debugger
- Find Replace
- Rule Relations
- Rule Trace
- Inference Browser



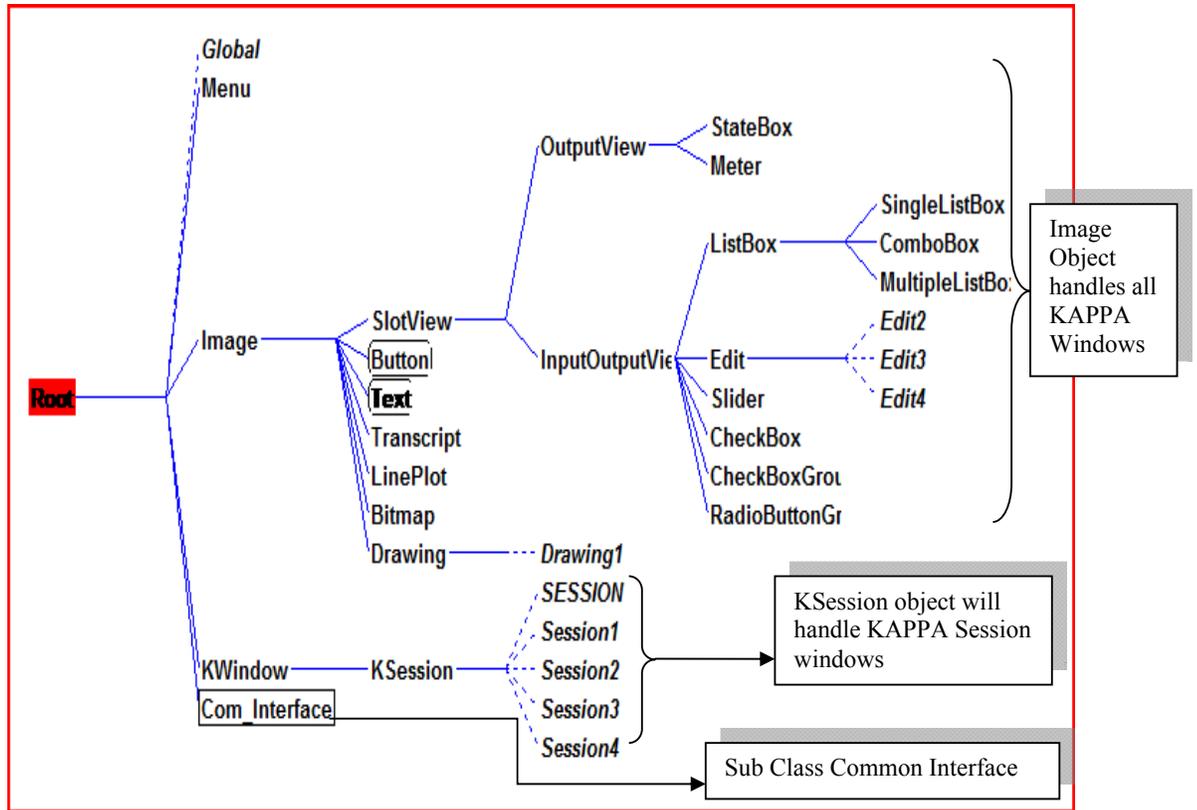
**Fig. 5.1: KAPPA Main Window with several Icons**

**5.2.1 Object Browser:** Object Browser is the most intuitive method of interacting with KAPPA. There are four main classes and a Global Instance in KAPPA (see fig. 5.2), namely Root Class, subclass Menu, sub class Image and subclass KWindow. Object Browser Editing menus may be used to create objects. The object editors are used to create methods and slots which define the behaviour and the properties of the object respectively. Figures 5.3 and 5.4 demonstrate how COIDS is connected to the KAPPA main object hierarchy.

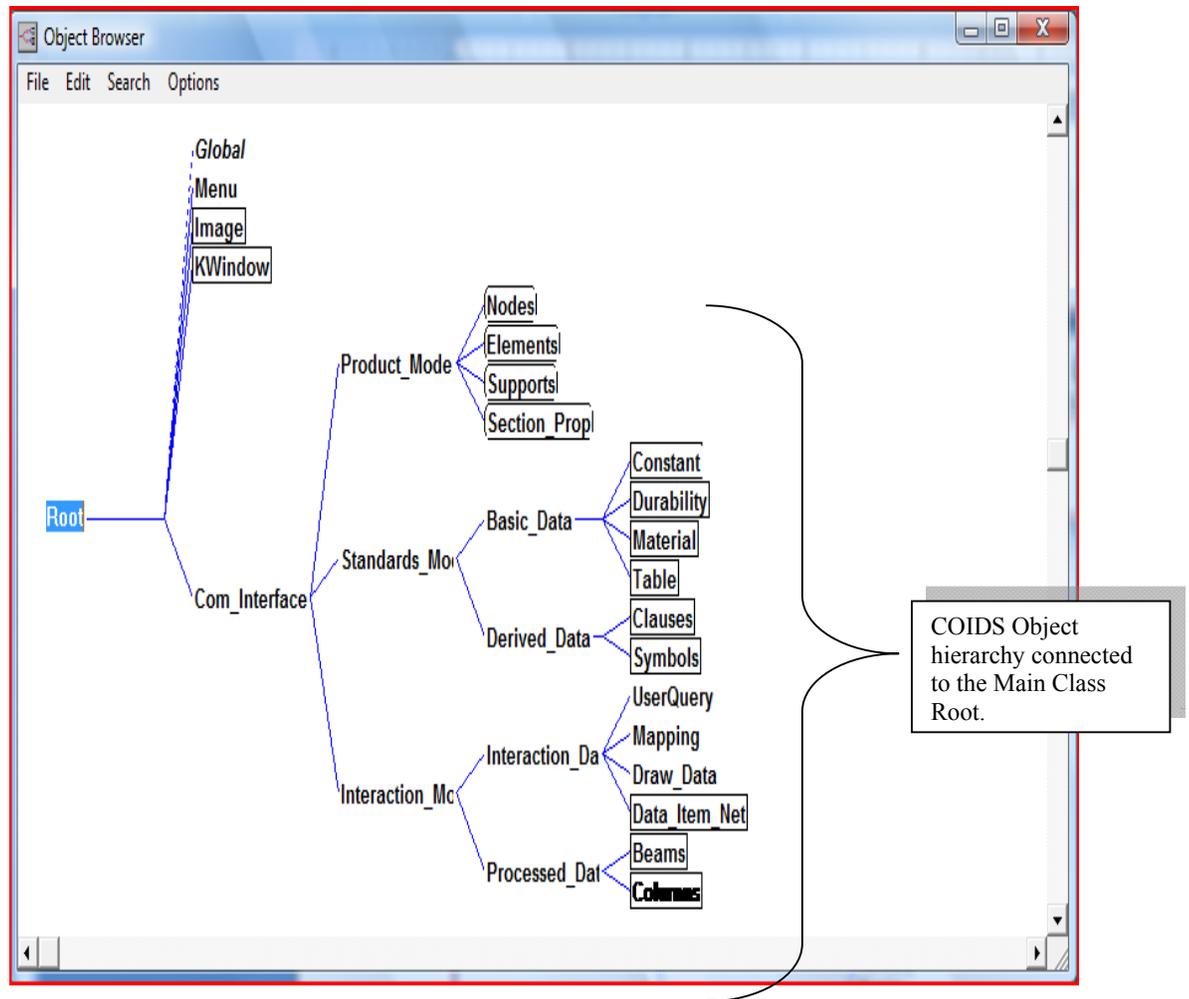
- Root Class: is the fundamental class from which all other classes in KAPPA are defined. It cannot be renamed or deleted. This class does not consist of any slots or methods (see Fig. 5.4).
- Menu Subclass: This subclass handles the KAPPA main menus
- Image Subclass: KAPPA images are handled by this class
- KWindow Subclass: All the session Windows are handled by this object



**Fig. 5.2: KAPPA's Main Object hierarchy**



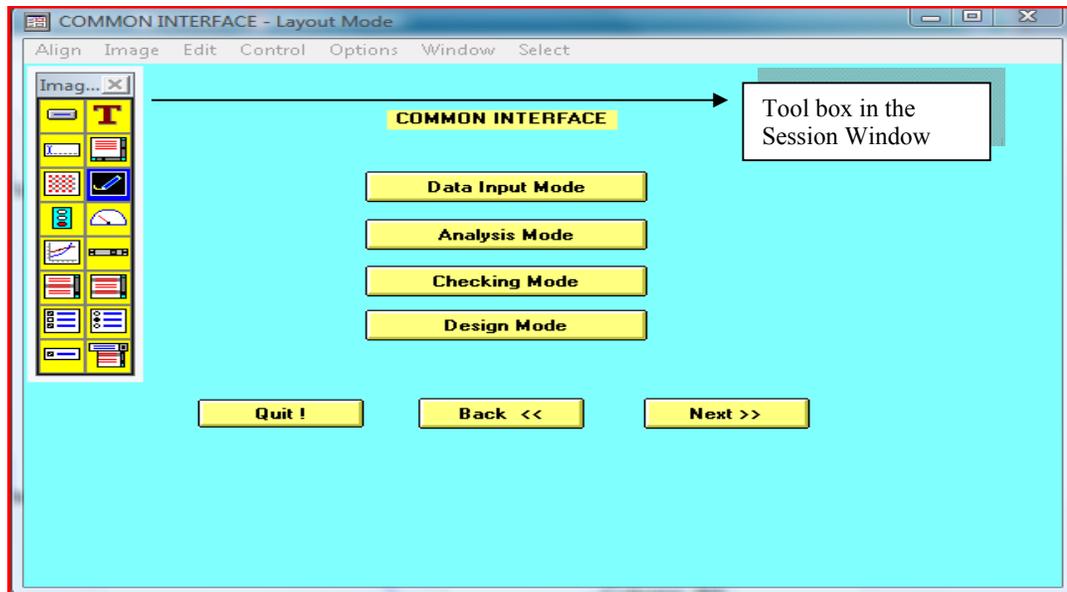
**Fig. 5.3: KAPPA's Extended Object Hierarchy which indicates the image object hierarchy**



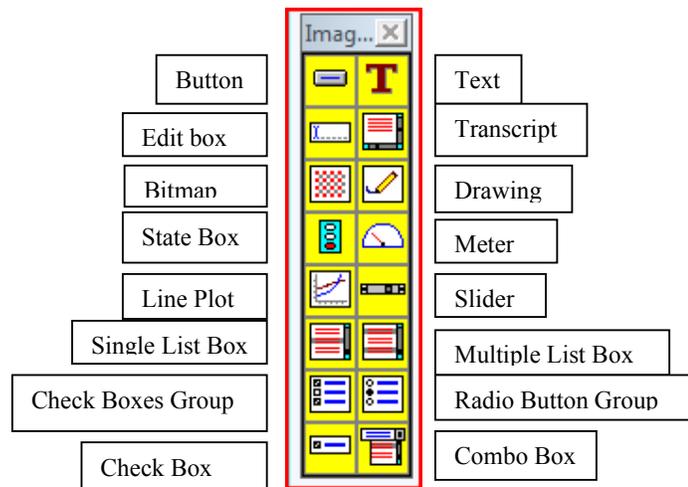
**Fig. 5.4: COIDS Object Connected to the Class Root**

**5.2.2 Session Windows:** KAPPA Session windows are very important tools to develop the graphic user interface of the knowledge base. These Image tools will help the user to interact with the knowledge base. These tools will also be used to display the output result of the complex knowledge base.

These windows are provided with Active Image tool boxes (see fig. 5.5). This tool box package contains Buttons, Edit box, Bitmap, State Box, Line Plot, Single List Box, Check Boxes group, Check box, Text, Transcript, Drawing, Meter, Slider, Multiple list box, Radio Button Group and Combo Box (see Fig. 5.6). Most of these active images are used in the development of COIDS graphic user interface.



**Fig. 5.5: KAPPA Active Images Tool Box**



**Fig. 5.6: KAPPA Active Images package**

**5.2.3 Editing Tools:** The Class and Instance Editor in KAPPA, represented as icons in the knowledge Tools window, provide a faster and more efficient means of creating, editing and saving the knowledge base. The Knowledge Tools window displays icons for the five principal editors in KAPPA, namely Class Editor, Instance Editor, Function Editor, Rule Editor and the Goal Editor (see Fig. 5.7).

- Class Editor can be used to create a new Class or edit, delete and rename a Class Object.

- Instance Editor can be used to create a new Instance or edit, delete and rename a Instance.
- Function Editor is used to edit the KAPPA Active Image package functions.
- Rule Editor is used to Edit rules that will be subjected to Forward Channing
- Goal Editor is used to edit rules to that will be subjected to Backward Chaining.



**Fig. 5.7: KAPPA Knowledge Tool Window**

### **5.3 Knowledge processing Techniques in KAPPA**

The knowledge base built in KAPPA needs to be accessed and executed. The knowledge in KAPPA is stored in objects as method or slot values. The execution of the KAPPA knowledge base is carried out by two separate inferencing processes: “Forward chaining”, activated by the Forward chaining function and “Backward chaining”, activated by the “Backward chaining” function. The graphic user interface is activated by KAPPA “functions”.

- Forward chaining attempts to discover a matching rule from the sets of rules in the agenda, by comparing the “If” part of the rule to the new fact. If the new fact matches the pattern of one of the conditions in the rule’s “If” part, then the chosen rule will be activated. Since standards clauses in COIDS are represented as methods in class Clauses, these methods are activated by KAPPA rules which are named by the same name as the corresponding clause.

For example the rule BS3444, states that if the check condition is rectangular beam for flexure according to the British Standards, check then evaluates clause BS 3.4.4.4 (see Fig. 5.8). The class Clauses slots hold the agenda or the list of clause numbers that need to be subjected for inferencing (see Fig. 5.9). For example if the effective span of a beam is to be evaluated according to British Standards, one of the following clauses BS 3.4.1.2, BS 3.4.1.3, BS 3.4.1.4 shall be evaluated, the criteria for selection of the particular clause being based on the support conditions, i.e. whether the beam is simply supported, continuous or cantilever (see fig. 5.9). This data is provided in the product model; thus when forward chaining the above clauses, the clause that matches the support condition will be picked and evaluated. A conventional program by contrast, has to indicate explicitly when given conditional statements should apply.

- In backward chaining, or goal driven reasoning, the inference engine tries to verify a fact (reach a goal) by finding rules that can prove the fact and then attempting to verify their premises. The premises in turn become new facts to be verified by other rules, and the process goes on. Backward chaining is appropriate when there is a specific question to be asked (that is, a specific goal to be reached); it is often used in diagnosis and classification applications. The KAPPA backward chainer attempts to verify a hypothesis by comparing the “Then” part of the rule. If the goal matches the pattern of one of the expressions in a rule’s “Then” part and if all the expressions in the rule’s “If” part are variables, the rule can apply. This means that the actions represented by expressions in the “then” part of the rule are taken. Typically, these actions add new information to the system in the form of slot values.

Every time a rule premise having to do with the value of a single-valued slot cannot be immediately verified during the backward chaining, the premise itself becomes a new goal to be resolved by further backward chaining. By repeating this cycle over and over until all goals are resolved, a rule chain is created that starts with one goal and ends by adding additional facts inferred by the rules. The backward chaining technique is also used to terminate a Forward chaining process.

COIDS uses a goal “CheckCondition” to find the stress state for which to check the element. Thus the backward chaining process will be continued until the goal is reached (see Fig. 5.10).

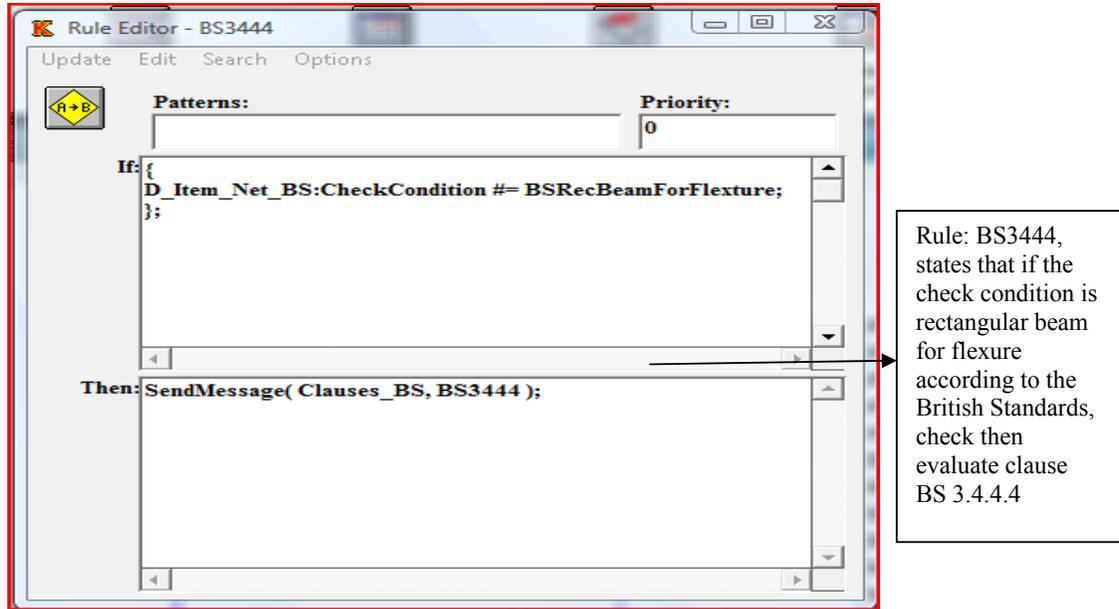


Fig. 5.8: KAPPA Rules linked to the COIDS standards clause BS3444

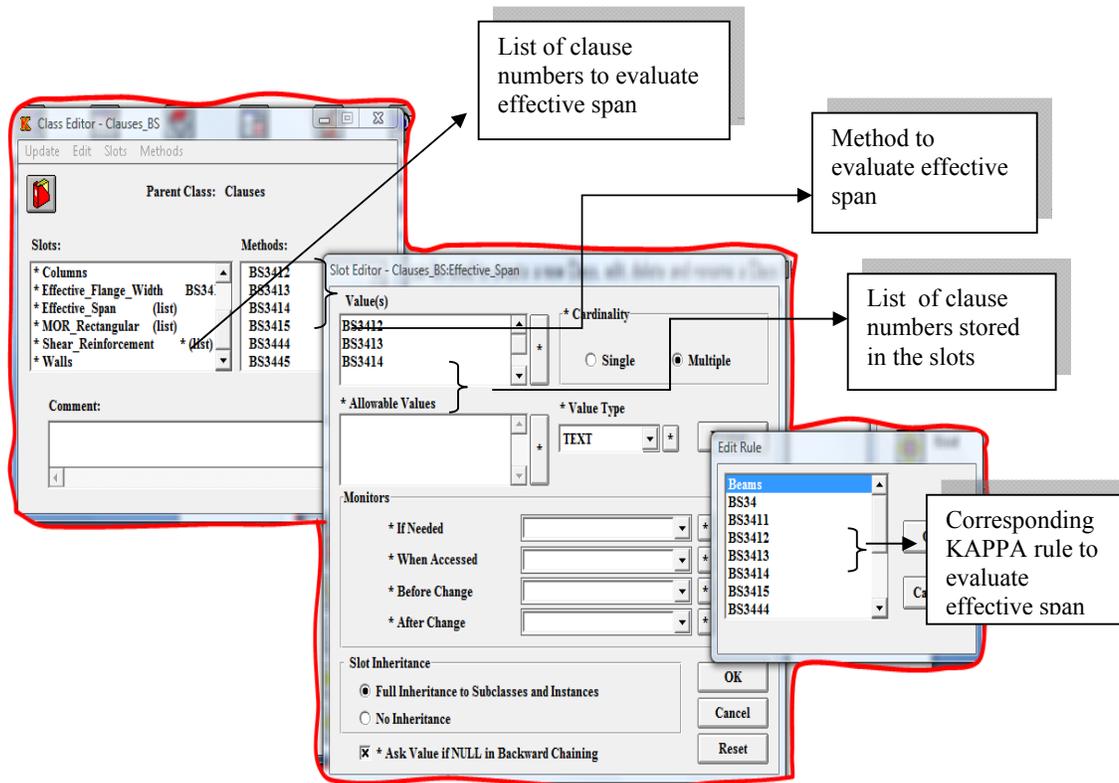
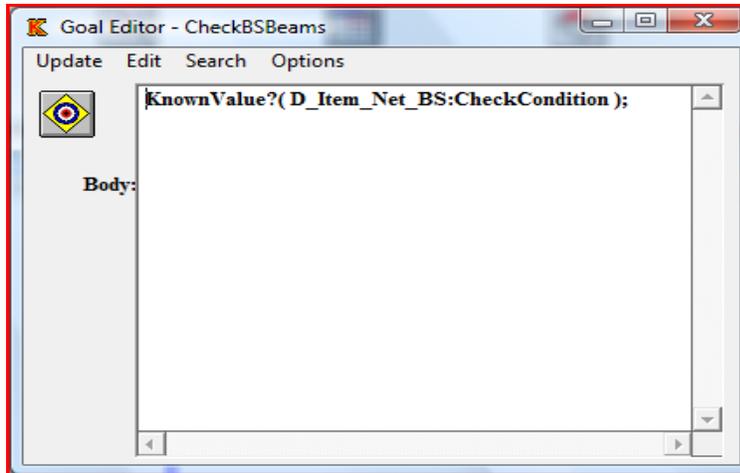


Fig. 5.9: COIDS strategy to execute standards clauses using KAPPA rules

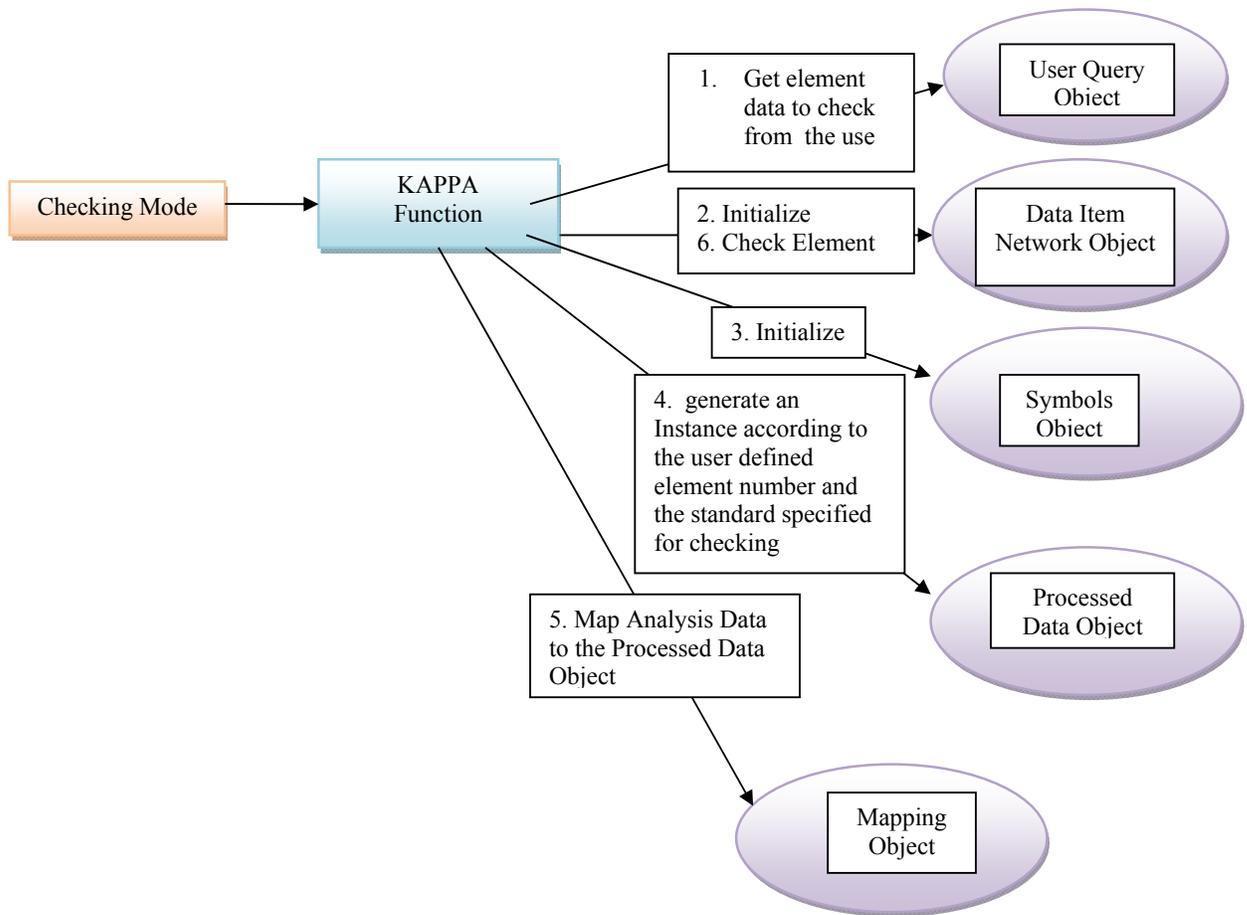


**Fig. 5.10: COIDS Goal “CheckCondition” to find the stress state of the element to be checked.**

#### **5.4 Execution process of COIDS in checking mode**

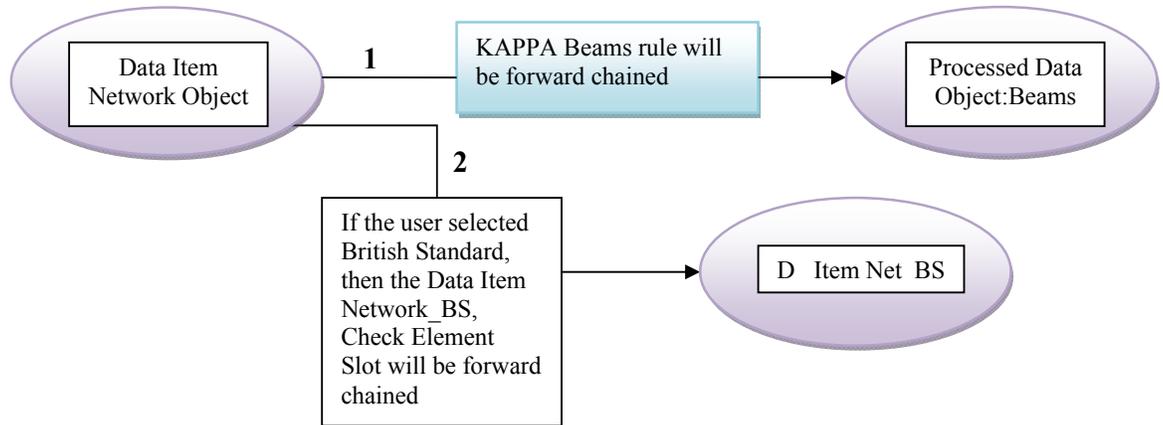
The standards processing procedure adopted by COIDS will simulate the procedure followed by a practicing engineer. This involves, first the selection of the element to be checked, then the selection of the standard that the element shall conform to, selecting the critical stress states to be checked and going through the relevant clauses of the standard to calculate the minimum reinforcement requirement and compare with the provided reinforcement.

When user presses the Checking Mode the following instructions will be generated from the KAPPA Functions to the COIDS objects as per Fig. 5.11. The subsequent message/ instruction flow between the user and COIDS is demonstrated by the following flow charts.



**Fig. 5.11: Instruction flow from KAPPA Functions to COIDS (Stage 1)**

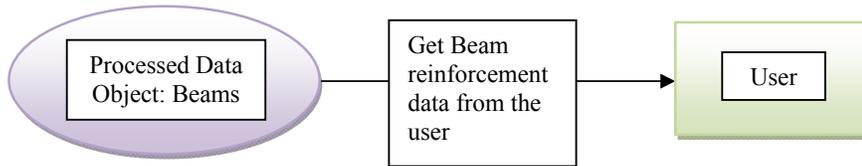
Assume the user decided to check a beam to British Standards; then the program will identify the element is a “Beams” and will be stored in the Data\_Item\_Network Object “CheckElement” slot. “CheckElement” method in the Data\_Item\_Network Object will specify to forward chain the “CheckElement” Slot value. When the KAPPA Functions sends the message to Data\_Item\_Network objects to execute the method “CheckElement”, since the slot will indicate “Beams” (because the user chosen element is a beam), KAPPA Beam rules will be forward chained (see Fig. 5.12).



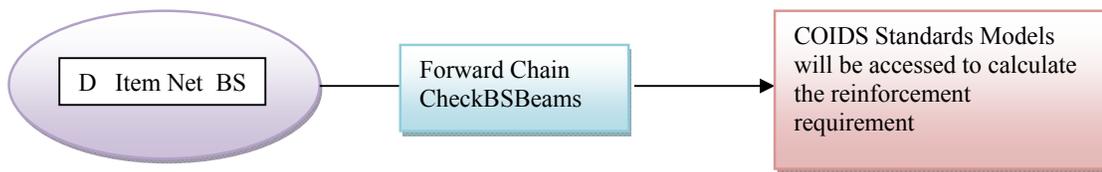
**Fig. 5.12: Data Item Network Object sends a message to forward chain the KAPPA rules (Stage 2)**

When forward chaining the KAPPA rule “Beams”, the rule will check whether the chosen element is a “beam” and send the message to the class Processed\_Data to get the beam type (rectangular), percentage redistribution and reinforcement data from the user (see Fig. 5.13). Here the user query was generated by the Processed Data Object, since the all the Processed Data (calculated reinforcement) will be stored in this object.

If the user selected standard is British Standards, then the D\_Item net\_BS, CheckElement slot will have a flag named CheckBSBeams selected and forward chained. Then the corresponding KAPPA rule CheckBSBeams will be forward chained. This process will activate the CheckBeamData Method in the Data\_Item\_Network\_BS. This method will calculate the required reinforcement specified by the standard, by sending messages to the COIDS Standard Model (see Fig. 5.14).



**Fig. 5.13: Forward Chaining KAPPA Rule Beams will execute the method Get\_CheckBeamData to get data from the user (Stage 3)**



**Fig. 5.14: Forward Chaining rule CheckBSBeams will calculate the requirement specified by the standard (Stage 4)**

## CHAPTER 6

### TYPICAL COIDS SESSIONS

This chapter will demonstrate a typical COIDS session. The step by step approach will highlight the concepts described earlier. Graphic User interfaces (GUI) of COIDS will facilitate the user to interact with COIDS. There are three main session windows for the user, namely the Common Interface, Basic Data Input and Interaction Window. Common Interface session window is called the main window (see Fig. 6.1), since the all the operating modes such as Data Input Mode, Analysis Mode, Checking Mode and Design Mode (not implemented) are initiated by this session window. Activation of the modes will lead the user to subsequent session windows and to the dialog boxes generated by COIDS.

#### 6.1 Data Input Mode

This mode will assist the user to input the product data to COIDS. The Data Input Mode button will bring up the Basic data input session window (see Fig. 6.2).

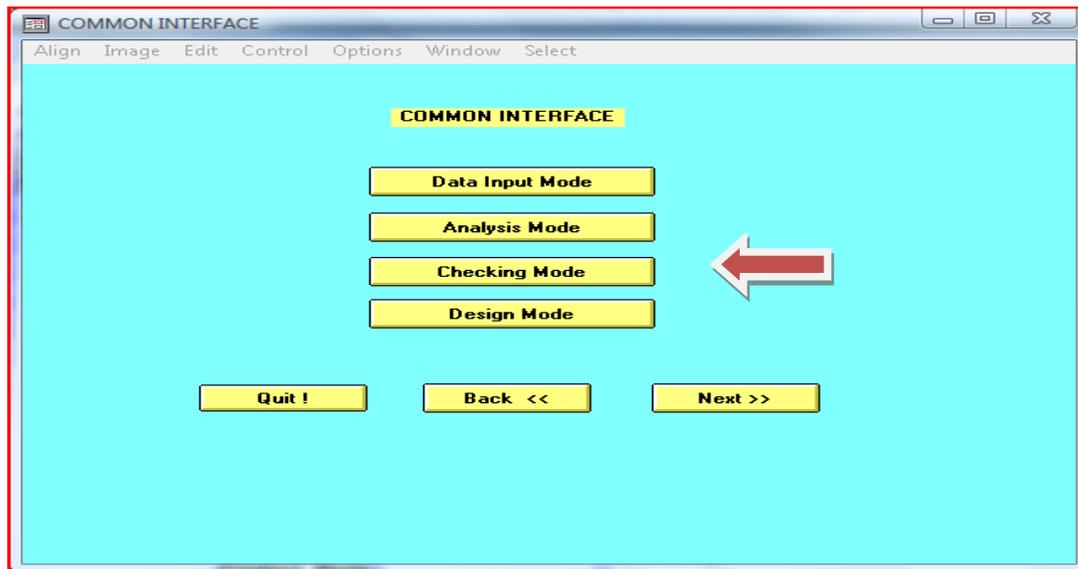
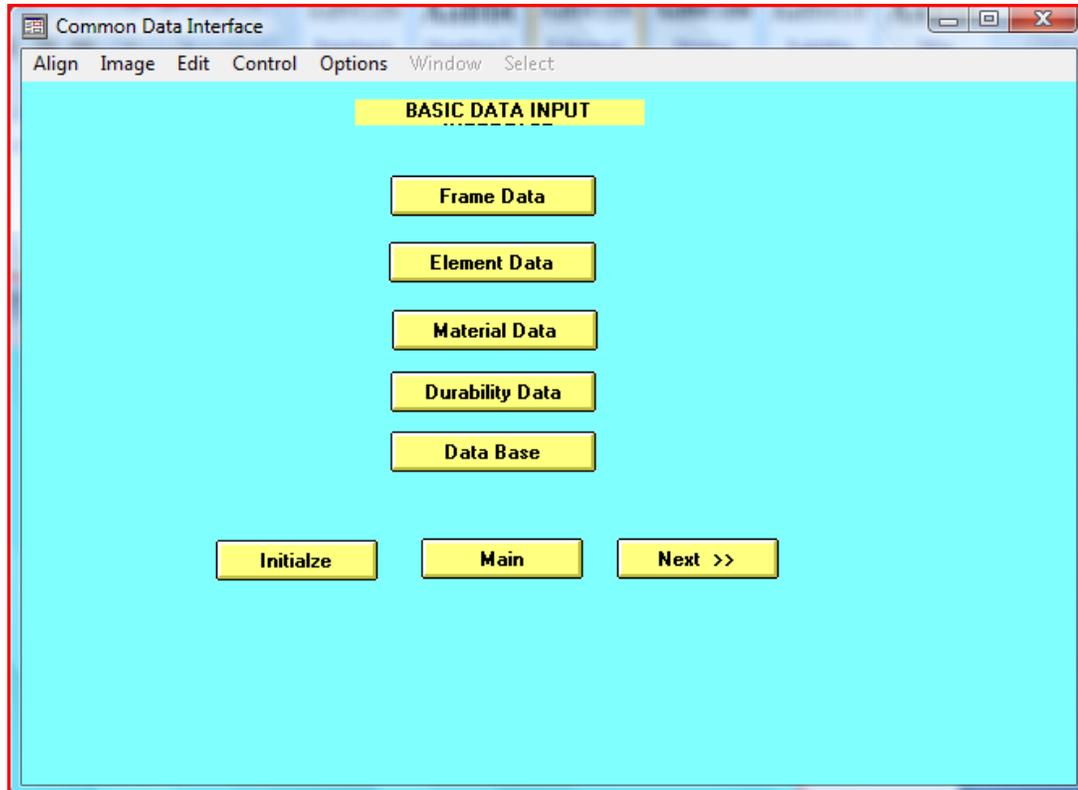


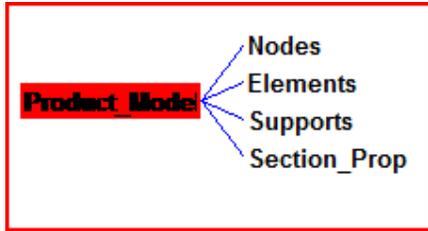
Fig. 6.1: Common Interface Session Window



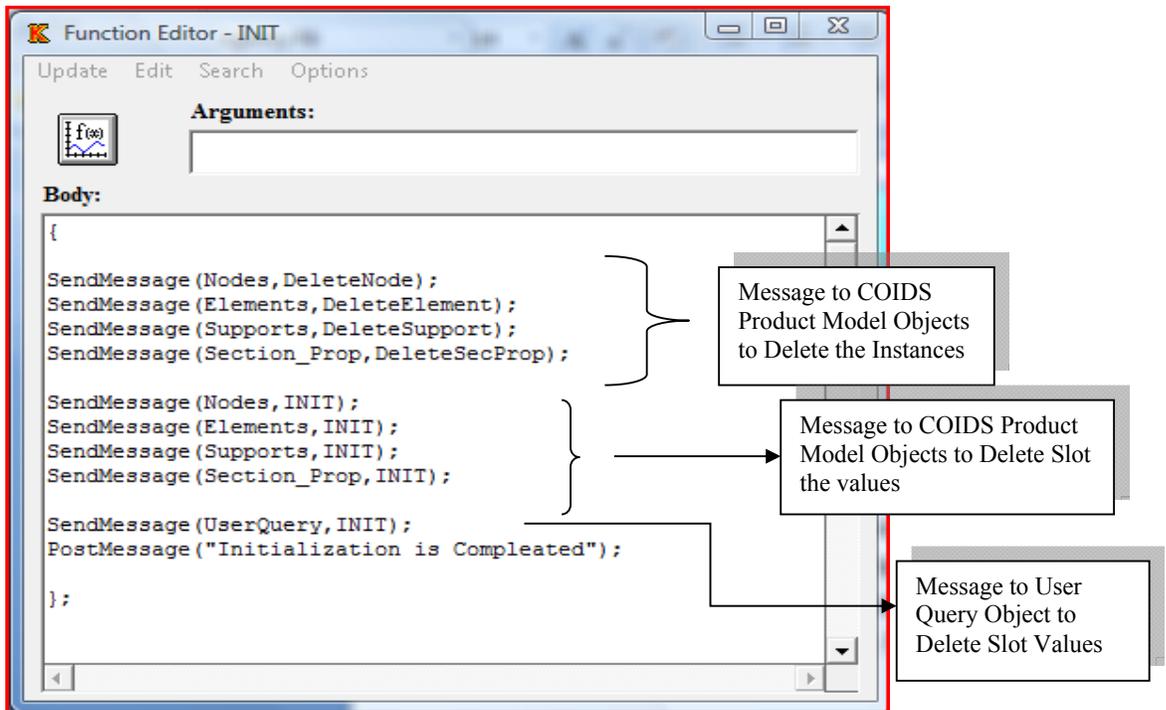
**Fig. 6.2: Basic Data Input Session Window**

The Basic Data Input mode will assist the user to input the product data such as Frame Data, Element Data, Material Data and Data Base Location (not implemented).

The first step of the user session will be to initialise the Product Model hierarchy (see Fig. 6.3), which is to delete all the instances of class Nodes, Elements, Supports and Section Properties. The KAPPA function editor's initialise method "INIT" (see Fig. 6.4) will send a series of messages to COIDS objects to delete slots and instances. This action will also delete the slot values of the parent classes and delete slot values of the Class UserQuery to accommodate the new user input. After initialising, this method will post a message to the user stating that the initialisation is complete (see Fig. 6.5). The "INIT" method demonstrates a key Object Oriented Programming (OOP) concept called "polymorphism", which means that the same message will perform different task in different objects.



**Fig. 6.3: KAPPA Product Model Hierarchy**

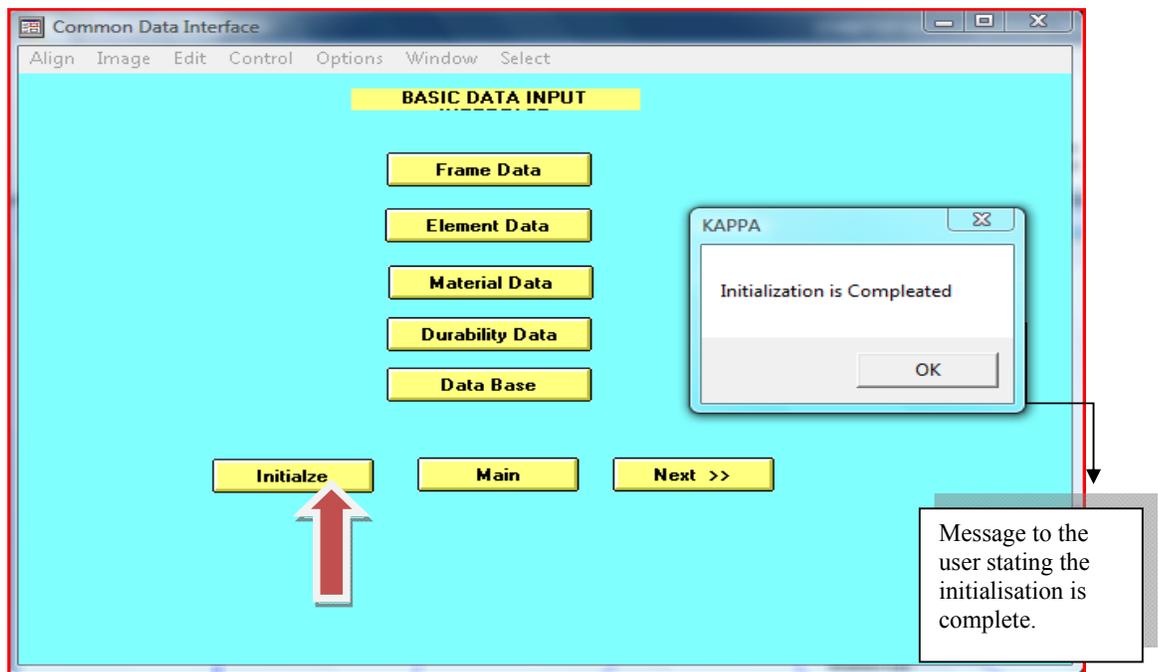


**Fig. 6.4: KAPPA Function Editor “INIT” method will send messages to COIDS Objects**

After Initialising the Product Model, the Frame Data button of the Basic Data Input Session Window can be activated. The Basic Product Data Dialog Box will be posted to the user (see Fig. 6.6).

The user inputs frame type, number of nodes, number of elements, number of supports and number of section properties that will be used to generate the Product Model instances (see Fig. 6.7).

The next step is to input data to the Object Instances such as Node Data, Element Data, Support Data, Section Properties, Element Loads and Nodal Loads. The Figures 6.8 to 6.17 illustrate the typical user input sessions. Material Data is the next input data to COIDS (see Fig 6.18 and Fig. 6.19). The Durability Data Inputs such as fire resistance and the exposure conditions can be input to the code by activating the Durability Data button in the Basic Data Input session window (see Fig. 6.20 and Fig. 6.21). The product data will be used to generate the Input Data File to the Analysis Package, in our case MFEAP, using the COIDS Class Mapping.



**Fig. 6.5: Basic Data Input Session Window, Initialise button will initialise the COIDS**

**INPUT DATA**

Job Name:

Frame Type:

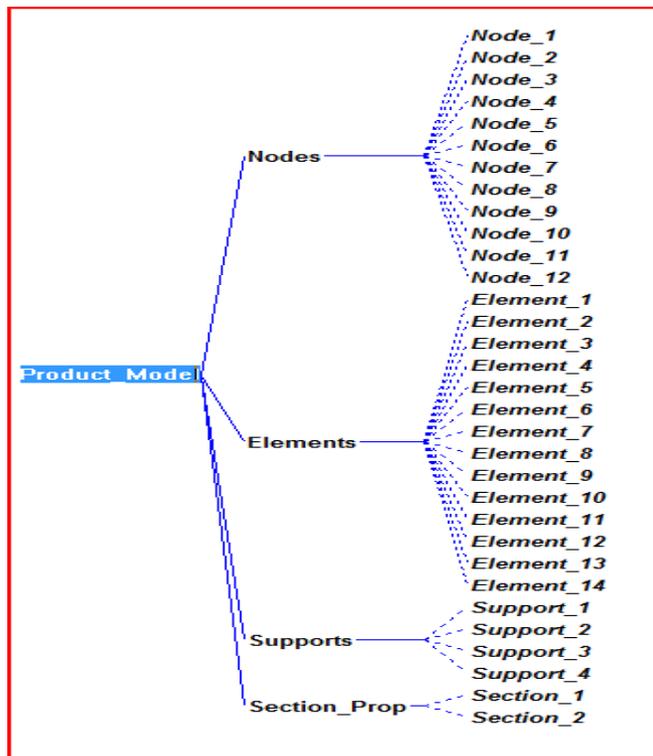
No. of Nodes:  2 ..

No. of Elements:  1 ..

No. of Supports:

Section Properties:  1 ..

**Fig. 6.6: Basic Product Data Dialog Box**



**Fig. 6.7: Product Model hierarchy with the Instances generated based on the frame data user input**

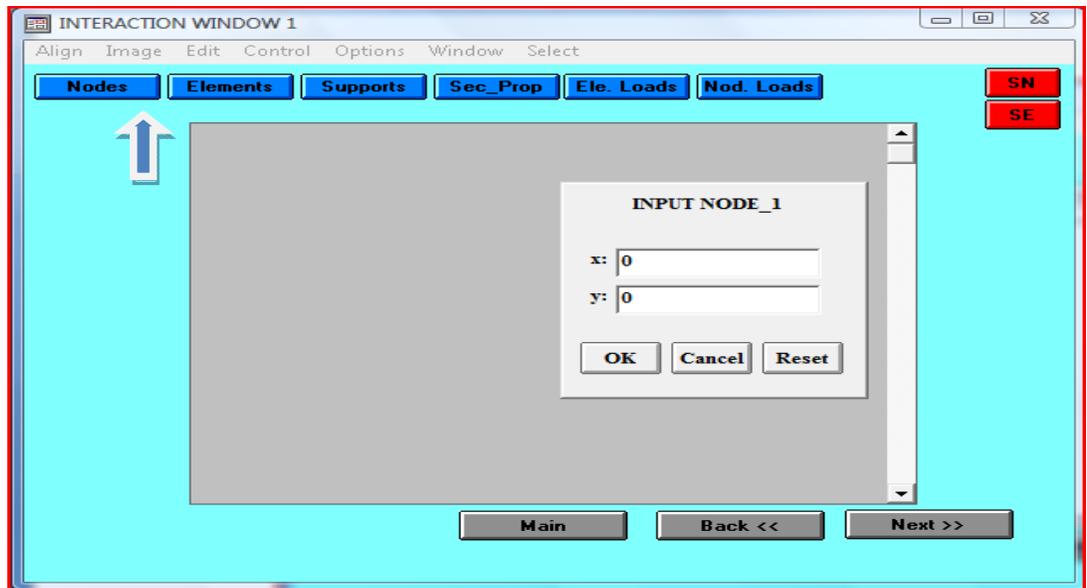


Fig. 6.8: Typical Node Data User Input

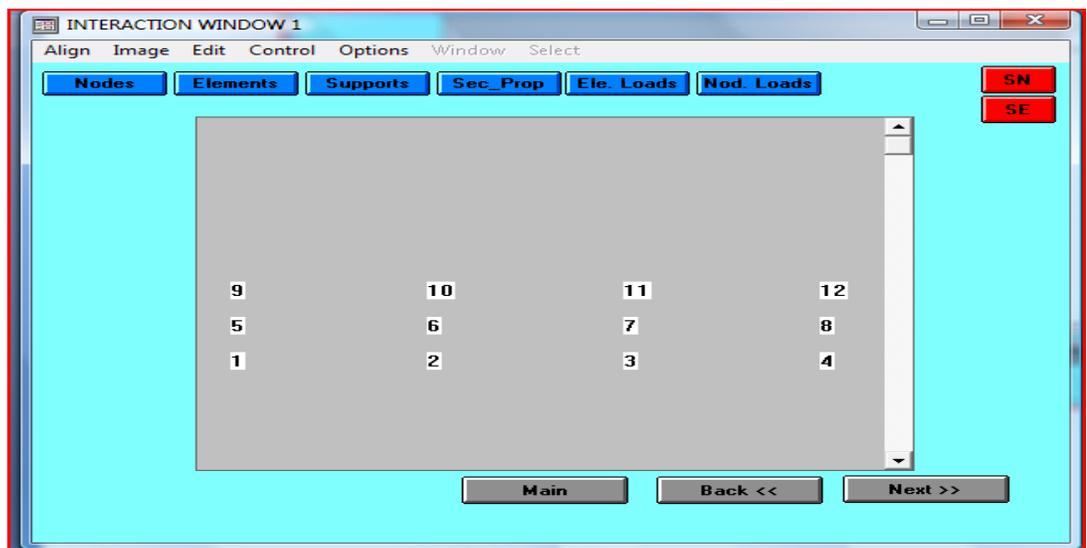


Fig. 6.9: Display User Node Inputs

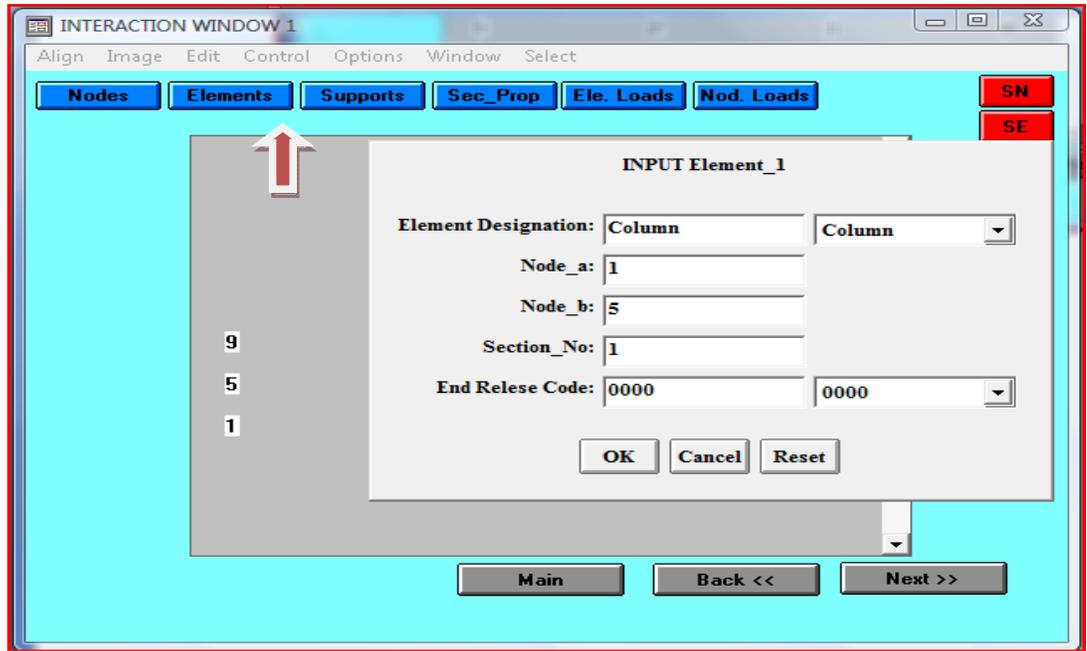


Fig. 6.10: Typical Element Data User Input

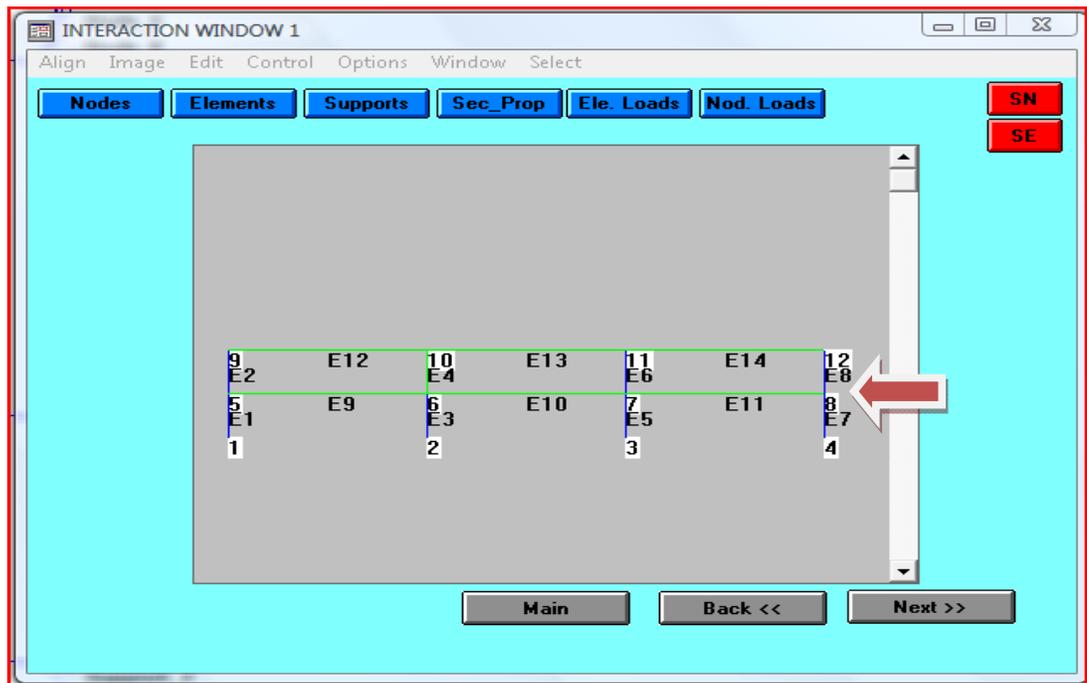


Fig. 6.11: Display User Element Inputs

INPUT Support\_1

Node Number:

Element Fixity:  XY ▼

Fig. 6.12: Typical Support Data User Input

INPUT SECTION\_1

height h(mm) :

width b(mm) :

Description :

Fig. 6.13: Typical Section Property Data User Input

User Request

Input Element Number

Fig. 6.14: Element number to input Element Load

User Request

Enter Number of Loads on the Element

Fig. 6.15: Number of Load types on the Element

User Request  
Please enter the value of Element\_9:Type\_1

UNIF

CONC
TRAP
<b>UNIF</b>

Comment...

Unknown

OK

Fig. 6.16: Load types on the Element to be defined

INPUT LOAD ON ELEMENT\_9

Uniform Load (kN/m): -10

Load Direction: FY

Load Type : Live

OK Cancel Reset

Fig. 6.17: Typical Uniform Load Input data

**User Request**  
Please enter the value of Material:MaterialType

Concrete

Concrete  
Steel

Comment...

Unknown

OK

Fig. 6.18: Typical Material Data User Input

**Input Material Data for Concrete**

Characteristic Cube Comp. Strength  $N/m^2$  25 20 .. 50

Tensile Strength  $N/m^2$  2

Density  $kN/m^3$  24

Poisson's Ratio 0.2

Elastic Modulus  $N/mm^2$  25E+06

OK Cancel Reset

Fig. 6.19: Typical Material Data User Input for concrete

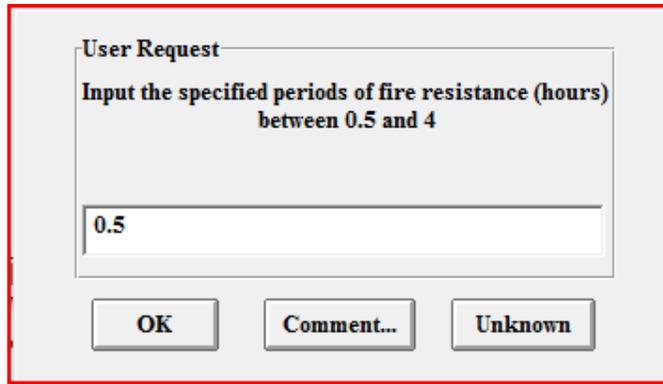


Fig. 6.20: Typical Durability Data User Input for Fire Resistance

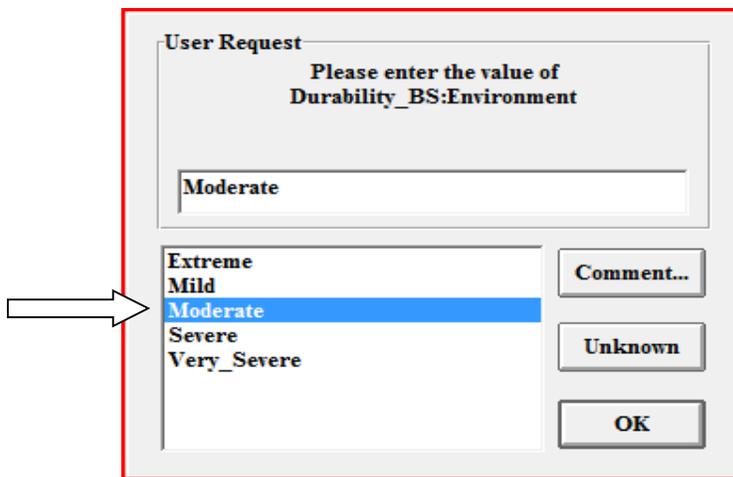


Fig. 6.21: Typical Durability Data User Input for Exposure Condition

## 6.2 Analysis Mode

The Product Data Analysis file will be used by the analysis package to analyse the input file. The Analysis Mode button of the main window will be used to activate the analysis session. COIDS will post a dialog box to define the path (location) of the executive file (exe file) of the Analysis Package (see Fig. 6.22). This user input will activate the analysis software (see Fig. 6.23), where the input data file could be accessed and processed into an output file. The processed file or the output file of the analysis session will be used by COIDS for conformance checking.

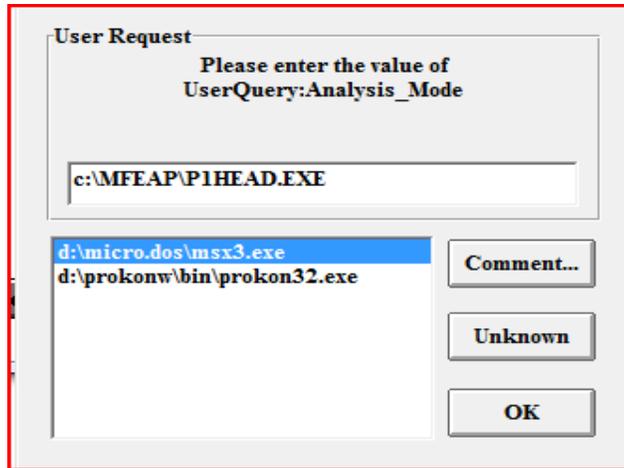


Fig. 6.22: User Define Path of the Analysis Package executive path

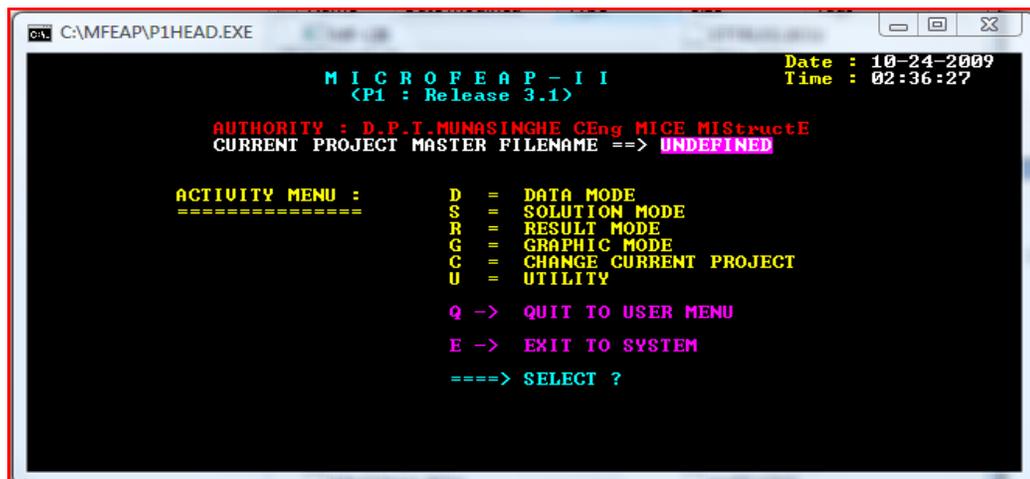


Fig. 6.23: MICROFEAP-11 Analysis Software

### 6.3 Checking Mode

The conformance checking of elements against different standards is the main task of COIDS. Checking mode of the Main Session Window will activate the checking process. Firstly, the user needs to input the element number to be checked (see Fig. 6.24). Then the user needs to specify which standard to be used in order to check the element (see Fig. 6.25). This action will generate the corresponding element instance in the Processed Data hierarchy under the relevant element type which has the specified standard tag (see Fig. 6.26). COIDS identifies the element type from the Product Data Information which is stored in the element instance. Then COIDS will post a dialog box to the user asking the user to insert the stress states such as Flexure,

Shear and Torsion or Serviceability Limit states such as Deflection and Crack width to be checked (see Fig. 6.27). Then the user needs to enter the path of the Analysis Data File in order to map the analysis data to the Processed Data Instance (see Fig. 6.28). After mapping the data to the relevant instance COIDS will post a message to the user stating that the task is complete (see Fig. 6.29). Where beams are concerned, after obtaining the element forces the user needs to input the percentage redistribution (see Fig.6.30) and reinforcement data for three critical sections (i.e., two supports and the middle section of the element). Here the section sizes are obtained from the product data and the user has the option to change if necessary - this is common in practice (see Fig. 6.31 to Fig. 6.33).

After completing all the input data for the checking stage, COIDS will compute the standards specified minimum reinforcement requirement and store this together with the user provided reinforcement in the slots of the element instance, e.g. Ele\_9 (see Fig. 6.34) . The user provided values will be compared with the standards specified required value and messages sent to the user stating that the section is “satisfactory” or “unsatisfactory” according to the user specified standard (see Figures 6.35 to 6.38).



**Fig. 6.24: User to insert the element number**

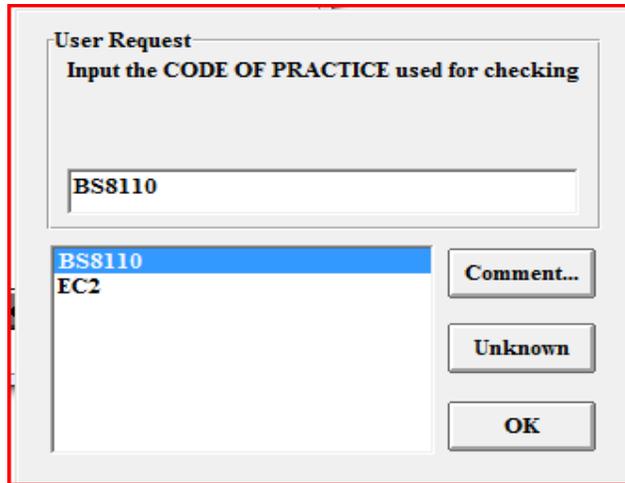


Fig. 6.25: User to insert which standard to be used for checking the element

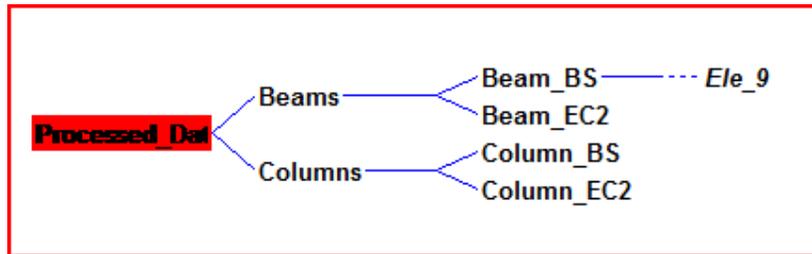


Fig. 6.26: Ele\_9 Instance is generated based on the user input

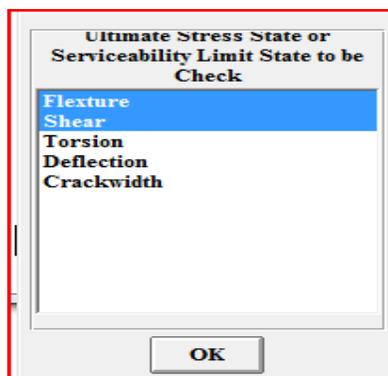
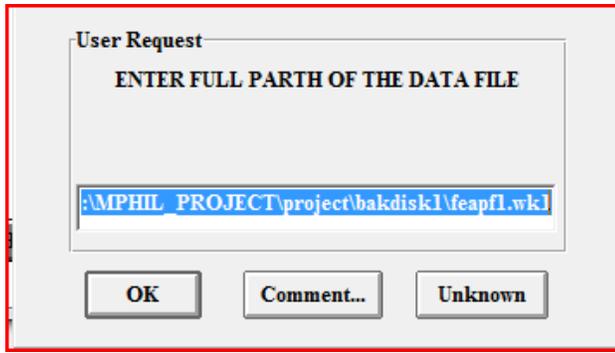
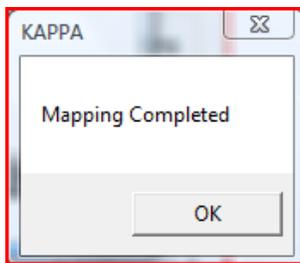


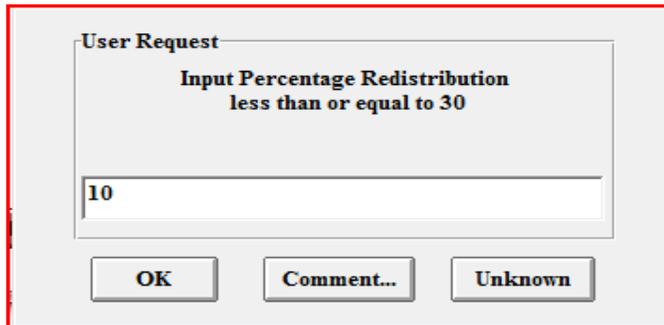
Fig. 6.27: User to input the stress states or Serviceability Limit States to be checked



**Fig. 6.28: User to input the path of the Analysis Data File**



**Fig. 6.29: Message to the user by COIDS indicating that Mapping is complete**



**Fig. 6.30: User to input the percentage redistribution**

**INPUT BEAM DATA - SECTION\_1**

Moment kNm

Section height h mm

Section web width b mm

Tension Steel

Compression Steel

Shear Links Spacing

Negative Moment  
(Hogging Moment)

Fig. 6.31: Reinforcement data at section 1 (Support)

**INPUT BEAM DATA - SECTION\_2**

Moment kNm

Section height h mm

Section web width b mm

Flange Shape

Maximum flange width b<sub>max</sub> mm:

Slab height mm

Tension Steel

Compression Steel A<sub>s1</sub>

Shear Links Spacing

Positive Moment  
(Sagging Moment)

Fig. 6.32: Reinforcement data at section 2 (Middle Section)

**INPUT BEAM DATA - SECTION\_3**

Moment kNm

Section height h mm

Section web width b mm

Tension Steel

Compression Steel

Shear Links Spacing

Negative Moment (Hogging Moment)

Fig. 6.33: Reinforcement data at section 3 (Support)

Instance Editor - Ele\_9

Update Edit Slots Methods

Parent Class: Beam\_BS

Slots:	Methods:
* As_cal1 431.959863	* BeamGen
* As_cal2 431.959863	* DeleteBeams
* As_cal3 431.959863	* DeleteMember
* As_pro1 402.119552	* DelSlot
* As_pro2 402.119552	* Element_No
* As_pro3 402.119552	* ElementType

Comment:

Calculated reinforcement area (mm<sup>2</sup>) requirement according to BS8110 (1985)

User Provided Reinforcement area (mm<sup>2</sup>)

Fig. 6.34: Ele\_9 Instance which included the processed data items

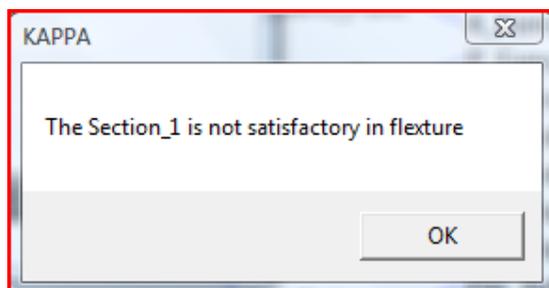
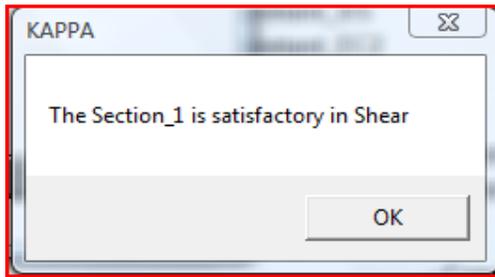
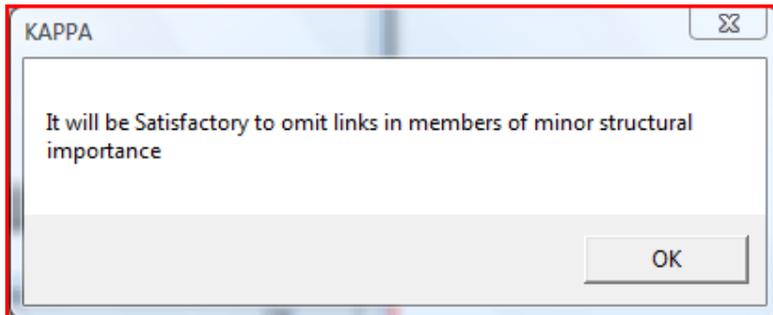


Fig. 6.35: COIDS message to user: Section 1 of the

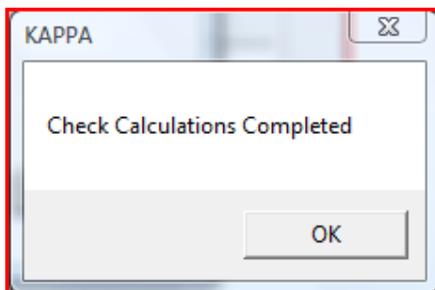
is satisfactory in Flexure



**Fig. 6.36: COIDS message to user: Section 1 is satisfactory in Shear**



**Fig. 6.37: COIDS message to user regarding shear reinforcement**



**Fig. 6.38: COIDS message to the user stating that the task is completed**

## **CHAPTER 7**

### **CONCLUSIONS AND RECOMMENDATIONS**

Although the objective of all reinforced concrete design codes is to provide specifications for safe structures, we have shown that there are wide variations in the actual provisions that lead to significant differences in design outputs.

It has been demonstrated that the Object Oriented Programming technique is able to represent reinforced concrete design codes in such a way that the structure of the code can be preserved and in a way that is faithful to the knowledge in and formats of code clauses and tables.

This has resulted in the standards knowledge not being hard coded into the programming code. Changes in knowledge arising from code updates can be easily made.

Inferencing techniques such as forward chaining and backward chaining were used to ensure that the design checking followed the procedure that would be adopted by a human expert.

The framework developed is also able to incorporate design checking to a number of codes, because the standards specific information is stored separately from the information regarding the structure. This concept has been termed “Common Interface for Design Standards” (COIDS).

Although the COIDS concept has been formulated, it has been tested using only a single standard, namely BS8110. The most immediate and relevant recommendation for future work is to test the concept with another standard. This would ideally be EC2, because Sri Lankan structural design practice is in the process of changing from BS8110 to EC2, and there would be many instances where structures designed to one code may need to be checked against the provisions of the other.

Another avenue for future work is to introduce some expert system features into the model based representation described here, so that the system is able to give explanations to users if and when queried.

## REFERENCES

American Concrete Institute, 2008. Building Code Requirement for Structural Concrete (ACI 318-08) and Commentary, An ACI Standard, Reported by ACI committee 318: ACI.

American Society of Civil Engineers, 2006. ASCE/SEI 7-0 Minimum Design Loads for Buildings and other structures:ASCE.

Bureau of Indian Standards, 1986. *IS 876: Part II –Loading Standards*: BIS.

Bureau of Indian Standards, 2000. Plain and Reinforced Concrete- Code of Practice (Fourth Revision): BIS.

British Standards Institution, 1996. BS 6399: Part 1 Loading for buildings Part 1, Code of practice for dead and imposed loading: BSI.

British Standards Institution, 1997. *BS 8110: Part 1 Structural use of concrete*: BSI.

British Standards Institution, 2000. BS5950: Part 1 Structural use of Steelwork in buildings- Part1: code of practice for design-Rolled and welded sections: BSI.

British Standards Institution, 1991. Eurocode 1: EN 1991-1-1:2002: Actions on structures – Part 1-1: General actions- Densities, Self-weight, imposed loads for buildings: BSI.

British Standards Institution, 1992. Eurocode 2: DD ENV 1992-1-1:1992: Design of Concrete Structures-Part 1: General rules and rules for buildings- (together with United Kingdom National Application Document: BSI.

DIN DEUTSCHES INSTITUT FUR NORMUNG E.V., 1978. *DIN 1045 Concrete and Reinforced Concrete, Design and Construction*. Translated by Henry G Freeman, 1978: German Standards (DIN-Normen).

Dias, W.P.S., 1998. *Graded Examples in Reinforced Concrete Design*. Bangkok: ACECOMS, Asian Institute of Technology.

Garrett, Jr., J. H., 1990. Application of Knowledge-Based Expert System Techniques to Standards Representation and Usage, *Building and Environment Journal*, 25(3), pp. 241-251.

Garrett, Jr.,H., 1990. Knowledge-Based Expert System: Past, Present, and Future. *IABSE Periodica 3/1990*,International Association of Bridge and Structural Engineers, June, pp. 21-40.

Garrett, Jr., J. H. and Hakim M.M.,1992. Object-Oriented Model of Engineering Design Standards. *Journal of Computing in Civil Engineering*, July, 6(3), pp. 323-347.

Harris, J. R and Fenves S.J.,1980. *Modeling of Standards: Technical Aids for their Formulation, Expression and Use*, Technical Report NBSIR 80-1979, National Bureau of Standards, Washington, DC.

IntelliCorp, Inc., 1990. *KAPPA User's Guide*.US: IntelliCorp, Inc.

IntelliCorp, Inc., 1990. *KAPPA Reference Manual*.US: IntelliCorp, Inc.

Kumar B. and Topping B.,1989. A Prolog- Based Representation of Standards for Structural Design, in. *Artificial Intelligence Tools and Techniques for Civil and Structural Engineers*: Civil-Comp Press, Edinburgh, UK, pp 165-169.

Kodagoda.N., 1997. *Object Oriented expert system to Represent a Design Code*. [Report] ( Report submitted to Civil Engineering Department, University of Moratuwa, April 1997).

Menzies J.B., Gulvanessian H., 1998, *Eurocode for Structural Loading*, <http://products.ihc.com/BRE-SEQ/ip1398.htm> (Last accessed 20 Dec. 2009): BRE Press

Moss R. and Webster R., (2004). EC2 and BS8110 Compared. *The Structural Engineer* 82(6), 16 March, pp. 33-38.

Neilson, A.I., 1989. *A Hybrid Approach to the Representation and Processing of Design Standards*. Glasgow: Ph.D Thesis, Department of Civil Engineering, University of Strathclyde, UK.

Narayanan, R.S., 1994. *Concrete Structures: Eurocode EC2 and BS 8110 Compared*. Avon: Longman Group UK Limited, UK.

Rosenman, M. A. and Gero, J. S. (1985). Design Codes as Expert Systems: *Computer-Aided Design*, 17(9), November, pp. 399-409.

Standards Association of Australia, 1988. *AS 3600 Concrete Structures*: SAA.

Standards Association of Australia, 1989. *AS 1170.1 Minimum design loads on structures, Part 1: Dead and Live Loads and Load combinations*: SAA.

Stahl, F.I., Wright, R.N. Fenves, S.J.&Harris, J.R. 1983. Expressing Standards For Computer-Aided Building Design. *Computer Aided Design*, 15(6).

Zhang, X.J. and Yao J.L.,1989. Tools for Expert System Development in Damage Assessment. In: *Computing in Civil Engineering: Computers in Engineering Practice, Sixth Conference sponsored by the Technical Council on Computer Practices of the American Society of Civil Engineers*, Atlanta, GA, September 11-13, 1989, American Society of Civil Engineers: US.

