Software engineering aspects of flexible structural analysis systems

Software Engineering Aspects of Flexible Structural Analysis Systems

Des aspects "software" pour des systèmes flexibles d'analyse des structures

Software-Aspekte von flexiblen Konstruktionsrechensystemen

FRITS C. DE WITTE
Research engineer
IBBC-TNO, Software-engineering dpt.
Rijswijk, Holland

GER M.A. KUSTERS
Research engineer
IBBC-TNO, Software-engineering dpt.
Rijswijk, Holland

SUMMARY

This paper stresses the need for and presents a flexible computer system for structural analysis. Especially in the field of Research and Development modifications should be easily to deal with. Guidelines for the implementation of such a flexible system include: Use a highly modular program architecture, separate logical and physical data structures and separate the control structure from the rest of the system. Two examples, related to non-linear FEM-analysis, are given to demonstrate that this approach, indeed, leads to a flexible system.

RÉSUMÉ

Dans cette publication l'importance et la réalisation du développement des systèmes flexibles pour l'analyse structurelle sont accentuées. Surtout dans un domaine de Recherche et Développement il est avantageux que les modifications peuvent être traitées facilement. Comment assurer le "software" à jour et vivant dans un tel domaine? La réponse donnée dans ce rapport est triple: appliquez une architecture de programme d'une conception bien modulaire, séparez des structures de data logiques et physiques, séparez la structure controlée du reste du système. On donne deux exemples, concernant la méthode des éléments finis, qui démontrent que cette méthode résulte dans un système flexible.

ZUSAMMENFASSUNG

In dieser Abhandlung werden die Bedeutung und die Realisierung der Entwicklung flexibler Konstruktionsrechensysteme betont. Besonders auf dem Gebiet der Forschung und Entwicklung müssen Veränderungen berücksichtigt werden können. Die Frage ist wie man die "software" lebendig und neuzeitlich halten kann unter solchen Umständen. Die Antwort in diesem Berichte bezieht sich auf drie Hinsichten: verwende eine weit in Modulen entwickelte Programm-architektur, trenne logische und physische Datastrukturen und trenne das Steuerungssystem von den übrigen Systemen. Zwei Anwendungsbeispiele in Bezug auf die nicht-lineare Finite-Elemente-Analyse werden dargestellt um zu erlautern dass diese Methode zu einem flexiblen Systeme führt.

1.0 INTRODUCTION

1.1 Environment Of Use

The aspects of software engineering described in this paper are very much related to R&D-user environments in general and that of the authors in particular.

The latter is the Software Engineering Dept. of a R&D institute in the field of Structural Analysis and Material Behaviour. As computer simulation nowadays is an essential part of this research the support of an engineering analysis system based on the finite element method (FEM) is necessary..

1.2 Requirements

As research is, by nature, related to non-standard problems the system should be well suited for modification and extension in order to achieve easy adaptation to the non-standard problems on hand. This requirement is called Flexibility.

The need for flexibility not only stems from R&D in material- and structural behaviour but also from developments in numerical analysis (new methods), software engineering (languages, operating systems) and hardware (memory sizes, mini- and micro-computers).

Flexibility guards the integrity and consistency of the system. Without that, maintenance costs and unreliability will frustrate its use.

With respect to the D of R&D an important requirement for software systems is Portability [1], because of the fact that results of research should be available on different hardware configurations, both for customers and in-house use.

Some additional requirements for application software in general and R&D-software in particular are Efficiency - i.e. a minor load for the hardware - and User-friendlyness - i.e. a minor load for the user - .

Some ten years ago we had to decide, considering the above-mentioned requirements, either to use one (or more) of the existing and commercial available FEM-software systems or to develop our own system. Mainly because of the fact that the existing systems were not flexible enough we decided to develop our own FEM-software package (called DIANA for DIsplacement method ANAlyser). In this paper we describe some Software Engineering problems, mainly with respect to the flexibility requirement, and how these were solved during the development of DIANA.

2.0 IMPLEMENTATION LANGUAGE

The first problem is the choice of the programming language. For DIANA FORTRAN-IV was chosen as the system implementation language mainly because of the general availability of compilers and programmers for this language. Unfortunately this language is not very suitable for the development of flexible software systems. The main lack's are:

- Not well suited for realisation of a modular systems architecture
- Poor data-management facilities

So special care had to be taken about these short-commings. Some aspects related to it will now be described.

3.0 SYSTEMS ARCHITECTURE

A modular architecture is very important for large software systems. Modularity means that specific functions in the system, for instance the solution of the system of equations, are performed in specific parts of the program, called modules. Each module "M" has a clearly defined function and further-more a communication 'interface' "i" to other modules of the system for exchange of data. This architecture is shown in fig. 1.

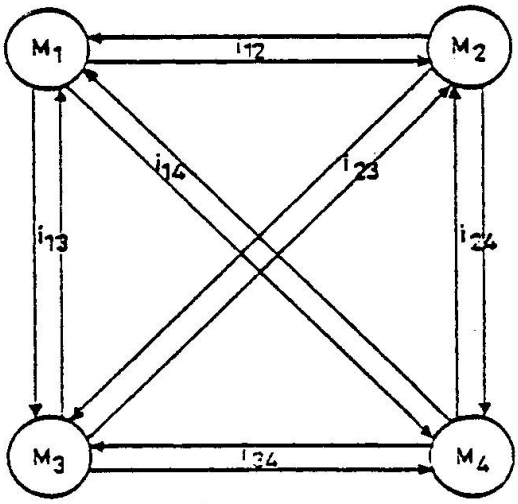

Figure 1

It is clear that the implementation of the performance of a module may be changed as long as the communication interfaces remain unaltered. It is possible, for instance, to change the solution method for the system of equations, without changing the other modules. This is the first step towards flexibility.

Problems with the architecture as shown in fig. 1 may arise when the number of modules increases. When each module of the system has its own interface to the other modules it is very difficult to connect new modules to the system. Many new interfaces have to be defined in that case.

A solution of this problem is to define one interface for the whole system and to connect all the modules to this interface. The interface separates the data from the modules of the system as shown in fig. 2.

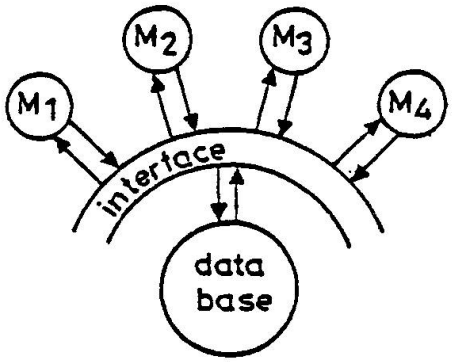

Figure 2

The data separated in this way may be called a database. The system modules communicate with the database via the interface, which may be considered as a database management system.

In DIANA we followed the approach as indicated in fig. 2. We will now describe the architecture of the DIANA system in more detail.

3.1 Modules

From the Users point of view the system is divided into Modules. Each Module performs a specific type of analysis. For instance:

- Module SOLVE for the solution of the system of equations.
- Module NONLIN for Physical and Geometrical non-linear analysis.
- Module SHOCK for dynamic analysis using direct time-integration.

The end-user activates a certain Module by a special command, followed by problem oriented Module-commands.

3.2 Segments

From the system's point of view Modules are built up from what are called Segments. Segments perform a specific function in a Module, for instance Segment DECOMP in Module SOLVE for decomposition of the system of equations. Each Module contains a special Segment for interpretation of the commands which are specific for that Module. For reasons of flexibility each Segment is linked independently from all other Segments and is in fact a stand-alone program for the computer.

In Fortran-terms a Segment consists of a Main-program and a number of Subroutines (and Functions). These routines are functionally collected in two groups:

1. Routines to perform a task specific for the Segment.

2. More general routines, to perform tasks which may be applicable in various Segments. These routines are assembled in Service Libraries, which are specified by the programmer during the Link-process of each Segment. For instance Service Libraries do exist for:
   - Data-management
   - Matrix end vector multiplication
   - Text manipulation.

The architecture described garantees a highly flexible software tool for R&D in the field of FEM-analysis. The standard versions of the Segments may be used as a tool-kit for FEM-programmers. However, they may adapt the system to their own needs by creating experimental versions of one or more Segments, without the need to Link the whole DIANA-system. A special command can be supplied, specifying the alternative version-name of the Segment. This feature is extremely usefull during development and implementation of new applications and theories in FEM-analysis.

Unfortunately some disadvantages are inherent to the concept of independently Linked Segments:

1. A large number of Segments have to be controled, i.e. to be loaded in the computer for execution in a specific sequence. This may need a large number of machine-dependent Job-Control statements, whereas special control structures as LOOPS and JUMPS may be even impossible on some machines.

2. Segments can only communicate via background storage. This may cause a large amount of data in the database which is very intensively accessed during the calculations. So special care has to be taken about the data management, both logical (for reasons of flexibility) and physical (for reasons of efficiency) .

In the next two sections we will describe how these two problems were

solved during the development of the DIANA-system.

4.0 CONTROL STRUCTURE

Two main aspects are related to the control of a DIANA-job:

1. The end-user likes to control the system by means of commands, related the problem he wants to solve (user--commands).

2. The DIANA-system can only control the loading and execution of the Segments (control-commands).

In some cases a conflict occurs between these two aspects. For instance, using the Module NONLIN to execute the next 5 loadsteps in a non-linear analysis the user would give a command like:

```
EXECUTE 5 LOAD STEPS
```

while the control of the Segments would be something like:

```
LOOP 5 TIMES
"LOAD SEGMENT INISTEP OF MODULE NONLIN"
"LOAD SEGMENT .......  OF MODULE SOLVE"
"LOAD SEGMENT ......."
 ........
END LOOP
```

In this case the user-command has to be interpreted and expanded into several control-commands. We now describe how this process works in the DIANA-system.

4.1 User Commands

The user activates the DIANA control-system by a single (machine dependent) Job-Control statement, like:

```
$RUN DIANA                    for VAX/VMS, or:

$$DIANA                       for HARRIS/Vulcan, etc.
```

Next he supplies a module command indicating the Module he is going to use:

```
*NONLIN                       for the Module NONLIN.
```

The DIANA control-system now activates the command interpretor Segment of the appropriate Module. This interpretor scans the module dependent user-commands including parameters etc. that follow the module command, for instance:

```
EXECUTE 5 LOAD STEPS
SIZE 1.2(5)                        "Five equal sized steps of 1.2"
PERFORM NEWTON-RAPHSON MI=10       "Max. 10 equilib. iterations"
```

Commands are interpreted and checked on correct syntax until the next module command or a special 'end' command:

```
*END
```

If no errors are detected the interpretor generates the necessary control-commands.

4.2 Control Commands

The generated control-commands are stored in the database. These commands exists of a list (the command-list), containing the names of the DIANA Segments to be loaded for execution. Special commands are generated if it is necessary to execute the Segments conditionally (JUMP) or more than once (LOOP, END LOOP). Some examples will be given later on.

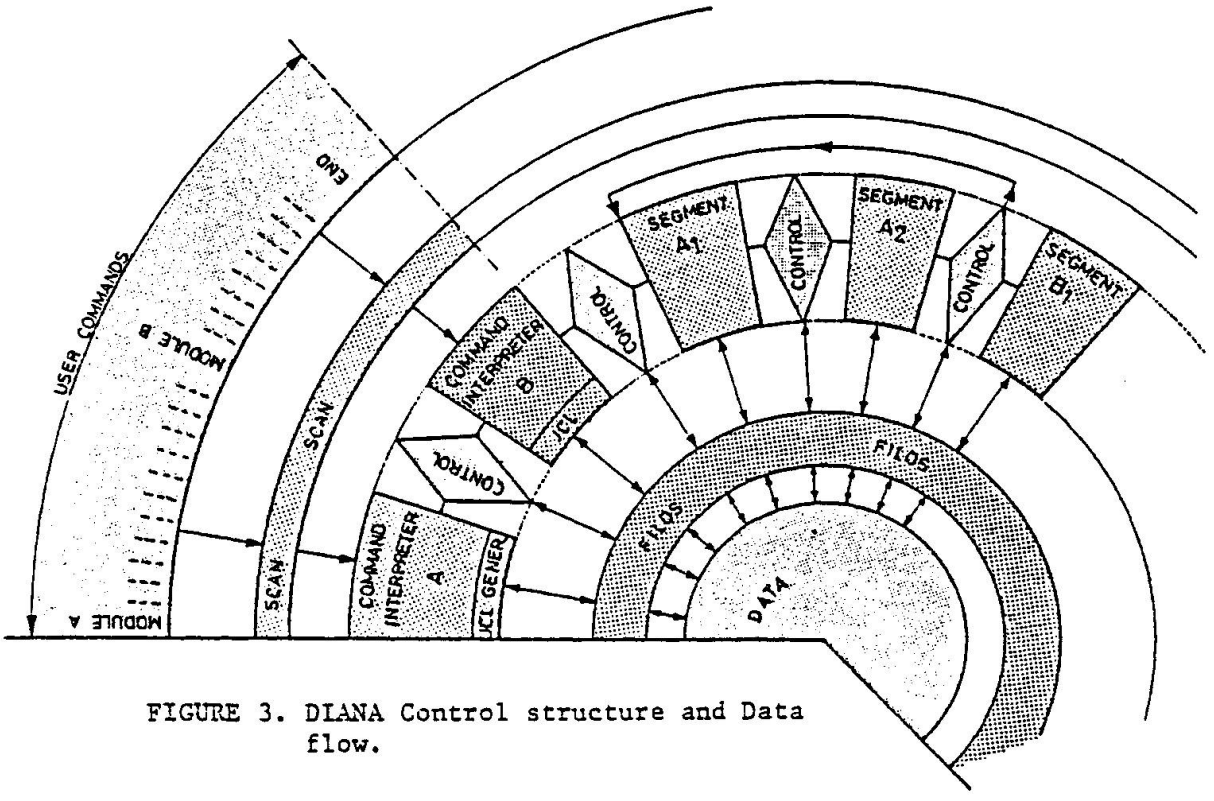

FIGURE 3. DIANA Control structure and Data flow.

When all user-commands are interpreted and all control-commands generated, the first Segment of the command-list is loaded for execution via a (machine dependent) Fortran-call.

After the execution of each Segment the DIANA control-system is activated to find out which is the next Segment to be executed. A special control-command (END) indicates the end of the command-list. The Control structure of the DIANA system is indicated in fig. 3.

The next section deals with data communication between Segments.

5.0 DATA MANAGEMENT

For a number of reasons much attention had to be paid to the organisation of the data management during the development of the DIANA--system:

- Complicated analysis of complex structures needs large amounts of data.
- Segments can only communicate via background storage (Disc--Files).
- The Fortran-IV language only gives a standard for sequential--access. Direct-access is often available, but is, especially in case of bufferd I/O, not portable.
- The Fortran-IV language does not provide for dynamic memory allocation.

Both portable and flexible data management can be realized using a "Database Management System". Unfortunately those systems generally are designed for administrative applications. Logical data-structures may be build for various kinds of access, but the physical organization of data in background memory is not tailored for minimal accestime.

For applications in an Engineering environment like DIANA the physical structure of the data in the database has to be designed for "High--volume I/O", i.e. minimal access-time is more important than minimal data-transport.

In DIANA therefore a severe distinction has been made between physical and logical data structures. The first are handeled by a dedicated service library, called FILOS for File Organisation System [6] . Although developed for DIANA, FILOS may also be used in other application software. It consists of a set of Fortran-callable routines to perform the following tasks:

- Data transport from foreground memory to background storage:
  - Data records may be accessed directly or sequentially, by name or by number.
  - All data may reside on one physical file, the database or FILOS-file.

- Organisation of data on the FILOS-file dedicated to minimal access-time. This is achieved by:
  - Buffering of Data-records, if appropriate.
  - Multiple buffering including a LRU-swapping algorithm.

- Dynamic memory allocation.
  Necessary Foreground memory is allocated at run-time, dependent on the size of the problem and the type of analysis.

FILOS acts as a machine independent, consistent interface to a physical data structure for Engineering applications. Logical data structures for dedicated applications may be given a physical representation using FILOS as a tool-kit.

In DIANA, for instance, logical data structures are designed for:

- Element data.
  Data records are accessed by name (STRESS, STRAIN etc.) for each element in the FEM-model. Data is organized in several levels (element, integration point etc.) and in several generations (Father-Son etc.).

- Material descriptions.
  Each material is described by a set of named data records.

- Node coordinates.

The logical data structures are very important for the flexibility of the DIANA-system. Each Segment may access, without major modifications, all the data stored by any other Segment of any other Module.

6.0 SOME APPLICATIONS

In this section we will demonstrate how the features of the DIANA-system described in the previous sections are used to implement different types of analysis methods.

6.1 Equilibrium Iteration Schemes

In this example we will show how the flexibility of the system is used to make different iteration procedures available in physical non-linear calculations. We will describe three different iteration schemes [2], [5]:

1. Constant stiffness matrix method, no updating of the stiffness matrix.

2. Modified Newton-Raphson, before each new loadstep the stiffness will be updated.

3. Normal Newton-Raphson, before each iteration the stiffness matrix is updated.

To perform a non-linear analysis with plasticity using the constant stiffness matrix method the user has to supply the following commands:

```
*NONLIN
   INITIALIZE
      ANALYSIS PHYSICAL GEOMETRICAL
      OPTIONS PLASTICITY CREEP
   END INITIALIZE
   EXECUTE nstep LOAD STEPS
      PERFORM CONSTANT MI=miter
      ANALYSIS PHYSICAL
      USE PLASTICITY
      SIZE r (nstep)
   END EXECUTE
*END
```

Where:

```
miter:  maximum number of iterations per loadstep.
nstep:  number of loadsteps to be performed.
r     :  size of loadstep increments.
```

The interpretor Segment generates the following command-list to perform a non-linear analysis with plasticity using the constant stiffness matrix method:

```
NONLIN INITIA              "Initialize data for non-lin. analysis"
LOOP nstep                 "Load-step loop"
   NONLIN LOAD             "Set up incremental load vector"
   SOLVE  SUBSTI           "Calculate incrementai displacements"
   NONLIN PHYSNL           "Determine internal load vector"
   LOOP miter              "Equilibrium iteration loop"
      SOLVE  SUBSTI
      NONLIN PHYSNL
   END LOOP
 END LOOP
```

To perform a non-linear analysis with plasticity using the Modified Newton-Raphson iteration method the user has to change the perform command in:

```
PERFORM MODIFIED NEWTON RAPHSON MI=miter
```

The interpretor Segment will generate the same command-list as previous but with some segments added at the beginning of the loadstep:

```
LOOP nstep
   NONLIN LOAD
   NONLIN TANGST          "Form new stiffness matrix"
   SOLVE  ASSEMB          "Assemble system of equations"
   SOLVE  DECOMP          "Decompose system of equations"
   SOLVE  SUBSTI
   NONLIN PHYSNL
   LOOP miter
      SOLVE  SUBSTI
      NONLIN PHYSNL
   END LOOP
END LOOP
```

The same segments added in the inner loop will result in the normal Newton-Raphson iteration procedure:

```
LOOP nstep
   NONLIN LOAD
   NONLIN TANGST
   SOLVE  ASSEMB
   SOLVE  DECOMP
   SOLVE  SUBSTI
   NONLIN PHYSNL
   LOOP miter
      NONLIN TANGST
      SOLVE  ASSEMB
      SOLVE  DECOMP
      SOLVE  SUBSTI
      NONLIN PHYSNL
   END LOOP
END LOOP
```

The example showed the flexibility of the system: by simply adding some Segments to the command-list different analysis methods became available, without the need to modify any of the used Segments.

6.2 Dynamic Analysis

In this example we will show how non-linear material behaviour can be included in a dynamic analysis using an implicit time integration method.

To perform a dynamic analysis, with implicit time integration the following equilibrium equations have to be solved [3], [4]:

$$M * {}^{t+\Delta t}\ddot{U} + C * {}^{t+\Delta t}\dot{U} + K * {}^{t+\Delta t}U = {}^{t+\Delta t}P - {}^{t}F \qquad \{1\}$$

Where:

M = mass matrix

C = damping matrix

^{t}K = tangent stiffness matrix at time t which includes the linear and non-linear material effects, and/or geometrical non-linear effects.

$^{t+\Delta t}P$ = externally applied forces at time $t+\Delta t$

^{t}F = nodal point forces vector equivalent to the stresses of the elements at time t.

$^{t+\Delta t}\ddot{U}$ = nodal point accelerations at time $t+\Delta t$

$^{t+\Delta t}\dot{U}$ = nodal point velocities at time $t+\Delta t$

$^{t}\ddot{U}$ = nodal point accelerations at time t

U = displacement increment from time t to time $t+\Delta t$

Solving equation (1) with the implicit Newmark Beta step by step integration procedure the following steps may be distinguished:

Initial calculations:

(I) Initialize:

^{0}U ; $^{0}\dot{U}$ and $^{0}\ddot{U}$

Calculate the constants a0 to a10 for Newmark Beta.

(II) Form effective coefficient matrix Ke:

$$Ke = K + a_0 * M + a_1 * C$$

In <u>linear analysis</u>:

(III) Triangularize effective coefficient matrix Ke (assemble and decompose).

For each <u>time step</u>:

(IV) Form effective load vector:

$$R^{t+\Delta t} = P^{t+\Delta t} + M * (a_0 * U^{t} + a_2 * U^{t.} + a_3 * U^{t..}) + C * (a_1 * U^{t} + a_4 * U^{t.} + a_5 * U^{t..})$$

(V) Solve for displacement increments:

$$Ke * U^{t+\Delta t} = R^{t+\Delta t} \; ; \quad U^{t+\Delta t} = U^{t+\Delta t} - U^{t}$$

(VI) Calculate new accelerations, velocities and displacements:

$$U^{t+\Delta t..} = a_6 * U^{t} + a_7 * U^{t.} + a_8 * U^{t..}$$

$$U^{t+\Delta t.} = U^{t.} + a_9 * U^{t..} + a_{10} * U^{t+\Delta t..}$$

$$U^{t+\Delta t} = U^{t} + U$$

To include non-linear material behaviour in the analysis the following steps are necessary per time step:

(III) Form new stiffness matrix K^{t}.

(IV) Form effective coefficient matrix Ke:

$$Ke = K^{t} + a_0 * M + a_1 * C$$

(V) Assemble and decompose effective coefficient matrix.

(VI) Form effective load vector:

$$R^{t+\Delta t} = P^{t*\Delta t} + M * (a_2 * U^{t.} + a_3 * U^{t..}) + C * (a_4 * U^{t.} + a_5 * U^{t..}) - F^{t}$$

(VII) Solve for displacement increments:

$$K_e * U = {}^{t+\Delta t}R$$

(VIII) If required, iterate for dynamic equilibrium:

(a) Initialize displacements U and inner iteration number i:

$$U = {}^{(0)}U ; \quad i=0$$

(b) i=i+1

(c) Calculate (i-1)st approximation to accelerations, velocities, and displacements:

$${}^{t+\Delta t..(i-1)}U = a_0 * {}^{(i-1)}U - a_2 * {}^{t.}U - a_3 * {}^{t..}U$$

$${}^{t+\Delta t.(i-1)}U = a_1 * {}^{(i-1)}U - a_4 * {}^{t.}U - a_5 * {}^{t..}U$$

$${}^{t+\Delta t\ (i-1)}U = {}^{(i-1)}U + {}^{t}U$$

(d) Calculate (i-1)st effective out-of-balance loads:

$${}^{t+\Delta t\ (i-1)}R = {}^{t+\Delta t}P - M * {}^{t+\Delta t..(i-1)}U + C * {}^{t+\Delta t.(i-1)}U - {}^{t+\Delta t\ (i-1)}F$$

(e) Solve for i-th correction to displacement increments:

$$K * {}^{(i)}\Delta U = {}^{t+\Delta t\ (i-1)}R$$

(f) Calculate new displacement increments:

$${}^{(i)}U = {}^{(i-1)}U + {}^{(i)}\Delta U$$

(g) Check for convergence, if not return to (b) for next iteration, otherwise continue:

$$U = {}^{(i)}U$$

(IX) Calculate new accelerations, velocities and displacements:

$$\overset{t+\Delta t}{U} = a_6 * \overset{t}{U} + a_7 * \overset{t}{U} + a_8 * \overset{t}{U}$$

$$\overset{t+\Delta t}{U} = \overset{t}{U} + a_9 * \overset{t}{U} + a_{10} * \overset{t+\Delta t}{U}$$

$$\overset{t+\Delta t}{U} = \overset{t}{U} + \overset{t}{U}$$

To perform a dynamic analysis with linear elastic material behaviour the user has to supply the following commands:

```
*SHOCK
   INITIALIZE dtime
      ANALYSIS NEWMARK alfa beta
   END INITIALIZE
   EXECUTE nstep TIME STEPS
      PERFORM LINEAR
   END EXECUTE
*END
```

Where:

dtime:	time increment.
alfa :	constant for Newmark Beta.
beta :	constant for Newmark Beta.
nstep:	number of time steps to perform.

The interpretor Segment generates the following command-list to perform a dynamic analysis with linear elastic material behaviour for nstep time steps (the number in brackets indicate the previously mentioned part of the integration procedure):

```
SHOCK INITIA          (I)
SHOCK EFCOEF          (II)
SOLVE ASSEMB          (III)
SOLVE DECOMP          (III)
LOOP nstep
   SHOCK EFFLOD       (IV)
   SOLVE SUBSTI       (V)
   SHOCK VELACC       (VI)
END LOOP
```

To perform a dynamic analysis with non-linear material behaviour and/or geometrical non-linear effects the execute command block has to be changed in:

```
EXECUTE nstep TIME STEPS
   PERFORM NEWTON RAPHSON MI=miter
   ANALYSIS PHYSICAL GEOMETRICAL
   USE PLASTICITY
END
```

The interpretor Segment generates the following command-list to perform a dynamic analysis with non-linear material behaviour and/or geometrical non-linear effects for nstep time steps:

```
SHOCK INITIA          (I)
LOOP nstep
   NONLIN TANGST      (III)
   SHOCK  EFCOEF      (IV)
   SOLVE  ASSEMB      (V)
   SOLVE  DECOMP      (V)
   SHOCK  EFFLOD      (VI)
   SOLVE  SUBSTI      (VII,VIIIa)
   NONLIN PHYSNL      (VI) F-term for i+1 .
   LOOP miter         (VIIIb)
     SHOCK  OOBFOR    (VIIIc,d)
     SOLVE  SUBSTI    (VIIIe,f,g)
     NONLIN PHYSNL    (VIIId) F-term for i+1 or next time step.
   END LOOP
   SHOCK VELACC       (IX)
END LOOP
```

In this example we see once more the flexibility of the system. The only new Segment to be programmed in order to make non-linear material behaviour available, is the Segment OOBFOR in Module SHOCK. Whereas the Segment PHYSNL of Module NONLIN may simply be added as it is.

7.0 CONCLUSIONS

For application software, especially when used in a R&D environment where all kinds of technological developments take place, change is the major factor to deal with. To keep the software alive and up-to-date Flexibility is the most important requirement. Therefore three considerations have to be kept in mind during the development of application software:

1. Use a highly modular program architecture. This facilitates the maintenance of a consistent set of software, avoiding the danger of rigidity.

2. Separate the logical and fysical data structures. This leads to both flexible data management and efficient inter-modular data communication.

3. Design a special control structure. This gives a flexible control over the execution sequences of segments from various modules. Different analysis methods may be implemented simply by an assembly of control-commands.

That these considerations indeed lead to flexibility, is demonstrated by two examples, taken from the DIANA FEM-software system:

- The implementation of different iteration schemes in non-linear analysis.
- The inclusion of non-linear material behaviour in a dynamic analysis.

8.0 REFERENCES

[1] van Beinum, Gerlach E.; Tolman, Frits P. and de Witte, Frits C.: "Portability Aspects of DIANA, A large Engineering Analysis System", Proc. 4th EASIT-Conference on 'Problems and experiences with quality assurance and portability of software', Luxembourg, 12-13 May 1980.

[2] Zienkiewicz, O.C.: "The Finite Element Method", Mc. Graw-Hill, London, 1977.

[3] Bathe, Klaus-Jurgen and Wilson, Edward L.: "Numerical Methods in Finite Element Analysis", Prentice-Hall, Englewood Cliffs, New Jersey, 1976.

[4] Bathe, Klaus-Jurgen: "Static and Dynamic Geometric and material Nonlinear Analysis using Adina", MIT Cambridge Massachusetts, May 1976.

[5] Kusters, Ger M.A.: "Non-linear material behaviour of reinforced concrete using the Finite Element Method", TNO-IBBC report nr. BI-77-36/07.1.22110, March 1977 (in Dutch).

[6] "File Organisation System - FILOS, Users Manual, version 1.2", TNO-IBBC, 1981.

Leere Seite
Blank page
Page vide