6 CONCLUSIONS AND FURTHER WORK

In the present work a shell for solution of optimisation and inverse problems in conjunction with simulation programmes was presented. Emphasis was placed on the open and flexible structure of the shell, which makes it general with respect to the variety of problems to which it can be applied as well as the simulation systems with which it can be used.

The shell was combined with a general finite element system *Elfen* and applied to selected problems related to metal forming. This provided a good test for the adequacy of the shell concepts from the point of view that a complex simulation system was successfully utilised for solution of optimisation problems involving non-linear and path dependent responses, coupling of phenomena, frictional contact and large deformations. Experience justified the initial idea of the optimisation system consisting of a set of independent tools for solution of individual subproblems. The shell offers a framework for connecting such tools in a common system where they can be combined in the solution of complex optimisation problems. Once these tools are linked with the shell, the necessary interaction between them is established and they can be employed for the solution of optimisation problems as a part of a synchronised solution environment.

One of the basic guidelines in shell design was that it should not impose any a priori restrictions regarding the type of optimisation problems to which it is applicable. It was however assumed throughout that the shell will be applied to problems where evaluation of the objective or constraint functions (and eventually their derivatives) includes an expensive numerical simulation. This assumption allowed the file interpreter to be used as a user interface, thus the focus was on openness and flexibility of the interface rather than its speed.

The assumption regarding computationally demanding numerical simulations also dictates demands for the optimisation algorithms incorporated in the shell. In most cases the time needed for algorithm housekeeping operations is insignificant and the number of direct analyses necessary for finding the solution should therefore be regarded as principal the measure of effectiveness of the algorithm.

The shell design aims at building a general optimisation system applicable to a wide variety of problems. To make this goal achievable, the shell must provide a flexible framework for implementation and testing of new utilities, which must be accompanied by use in practice. Such a broad scope implies a number of demands, which can not be met instantly but are a matter of systematic long term development. The last section in this text is therefore devoted to assessment of possible further development of the shell. This assessment is based on appreciation of the current state and practical experience, which gives indication regarding which development tasks will be among the most important in the future.

6.1 Further Work Related to the Optimisation Shell

Development tasks can be divided in two groups. The first group includes developments related to the shell structure and concepts, while the second group includes development and incorporation of modules with given functionality.

Development of a complete open library is currently regarded as a primary development task. Such a library will enable incorporation of modules developed in different places. It must provide a condensed standard set of simple to use functions, which still enable sufficient interaction with constituent units of the shell. A large portion of the library has already been implemented and must be equipped by appropriate documentation. Other parts of the library will be developed simultaneously with introduction of additional functionality and final definition of additional concepts. Another important task is definition of rules for adding functions for direct access to module functionality to the shell library. The current arrangement anticipates access to module functionality through the user interface, while the framework for making this functionality available for direct use in other modules has not yet been set up.

The file interpreter will probably undergo substantial changes. Experience has shown that use of the current implementation is sometimes difficult and prone to errors. In order to suppress this deficiency, the syntax will have to be modified, probably towards the syntax of some common high level language such as C. This will require partial revision of the interaction between the interpreter, the expression evaluator and the variable system. This will also affect the syntax checker and debugger, which alone need some improvements to become a more reliable tool for detection and elimination of errors in command files.

A great deficiency of the shell is that it does not have a sufficiently general common system for processing and presentation of results. Such a system should enable, for example, the storing of the complete information about the optimisation path for later presentation in a standard way. This should be accompanied by appropriate visualisation tools. Currently the optimisation path can be written to a file by using the appropriate output command in the analysis block of the command file.

The need for a modular and hierarchical library of optimisation algorithms has been indicated by practical experience. The basis of such a library is currently being set up, while its development is a long term task which will be strongly affected by simultaneous experience gained by use in practice. One argument for development of such a library is the observation that more complex optimisation methods often incorporate more basic algorithms for the solution of subproblems. A well structured hierarchical library can therefore significantly facilitate development of increasingly sophisticated algorithms developed for special purposes. The need for development of special purpose algorithms is always present in an optimisation system such as the shell. A readily available example which supports this statement is establishing a recurrent interaction between an optimisation algorithm and a finite element simulation in such a way that a finer mesh is used in simulation as the solution of the optimisation problem is approached. This can save time since a coarse mesh is used far from the solution where high accuracy of simulation results is not crucial.

There is another argument which supports development of optimisation algorithms simultaneously with development of the shell. The resulting library will include some facilities which are usually not a part of existing optimisation libraries, but are important for incorporation in the shell system in compliance with its concepts. Such facilities will enable e.g. use of a common system for reporting errors and a common system for presentation of results.

Development of the optimisation library will induce the need for a suitable testing environment. By now algorithms were tested either on practical examples or on test problems defined through the command file. In the future this should be supplemented by a system of standard test functions pre-defined in the shell. Such a testing suite will make comparison of different algorithms for similar problems easier. The test problems will be designed with features which are expected to be difficult for specific algorithms.

Development of a general shape optimisation module will begin in the near future. It will increase the applicability of the shell because shape plays an important role in almost all branches of engineering design. This module will include tools for definition of parameter dependent transformations of shape and appropriate functions for transformation of discrete sets of points as well as continuous domains. Module functions will act on the geometrical level, therefore it will be possible to combine module functionality with existing functionality already implemented in individual simulation systems.

More specific tasks will be related to further effective utilisation of simulation systems. A direct interface with the finite element system *Elfen* has now been implemented. This interface is currently on a very basic level and includes access to programme data structures and basic control over its execution. Higher level functions will be added in the future, which will enable e.g. direct use of built-in post-processing capabilities such as integration of derived quantities over surfaces and volumes. Use of the programme for solution of optimisation problems will be facilitated by introduction of optimisation entities. These are objects, which include the definition of geometrical entities that are involved in definition of the optimisation problems, specification of data needed by the shell and specification of operations which will be performed on this data.

Use of the shell for practical purposes will in certain cases impose stronger requirements with respect to simplicity of use. Such requirements will be met by building templates, which will utilise the shell for solution of specific sets of problems. These templates will represent upgrades of the shell user interface by trading a certain level of generality for the required user friendliness. Various facilities will be employed in building such templates, e.g. high level special purpose commands implemented in the shell, portions of code for the shell file interpreter prepared for accomplishing pre-defined combinations of tasks, and pre and post processing utilities integrated in the simulation environment which will be utilised for the solution of problems.