# A Case Study Using the IMSE Experimentation Tool

Jane Hillston, Andreas L. Opdahl & Rob Pooley
Department of Computer Science, University of Edinburgh, Scotland
and SINTEF, University of Trondheim, Norway

**Abstract**

This paper presents a tool for creating and running experiments within a performance modelling environment, and a practical case study through which its key features are illustrated. The case study is concerned with optimising the load arrangement of a hospital information system. The paper describes the experiments created to support the case study in a textual and a graphical format.

Both tool development and evaluation work are being carried out in parallel as part of the IMSE project[1]. The paper presents this evaluation which resulted in the elicitation of further requirements on the tool. We also describe how many of these have been incorporated into the final tool design. The paper concludes with a description of the objectives for further work on the experimenter tool.

## 1 Introduction

The Experimenter, described in this paper, is an environmental tool at the core of the Integrated Modelling Support Environment (IMSE) [Hughes and Potier 1989]. The IMSE

[1]The Integrated Modelling Support Environment project (IMSE) is ESPRIT2 project 2143, funded by the CEC. IMSE reports can be requested from STC Technology Ltd., Copthall House, Newcastle-under-Lyme, Staffs, England.

project aims to exploit current workstation technology and build upon recent advances in the support of modelling activities, in order to produce an integrated set of tools to help performance modellers in their work. This represents the next step from the tools which have emerged in recent years offering increased support of modelling. Such tools usually require the user to be familiar with the modelling paradigm involved but not necessarily with the details of the underlying mathematics or programming language. There are many examples of such tools: RESQ, QNAP, COPE, HIT and GreatSPN. Further details of these and other tools can be found in [Potier 1985; Abu El Ata 1986; Puigjaner 1988].

The rôle of the Experimenter within IMSE is to allow the use of models to take place at a higher level of abstraction, hiding from the user the details of using various modelling tools and statistical packages. This gives even greater flexibility to the treatment of models and the analysis of systems.

The main aim of the project is to address the performance evaluation of computer and information systems, Computer Integrated Manufacturing (CIM) applications and communication systems, looking closely at the possibilities for incorporating performance evaluation and validation naturally in the design process. In order to ensure the applicability of the toolset developed and to provide progressive feedback to the tool designers part of the effort of the project is devoted to the development of several reference applications, using the tools of the environment. The case study described in this paper results from that work.

# 2 IMSE and the Experimenter

## 2.1 IMSE

In recent years many people have recognised the desirability of emphasising and exploiting the similarities between system design and model construction [Pooley 1989; Zeigler 1984], and the advantages which would be gained if the two processes could be integrated to some extent. Such an integration would represent the first step towards incorporating

performance evaluation and validation naturally into the design process.

Within IMSE the tools supported are designed to assist in all stages of the modelling aspect of performance evaluation, from model construction and generation of workloads, through experimentation, to the generation of final reports on findings. Wherever possible the environment has been designed to support users without in-depth knowledge of the tools and techniques employed.

The tools are made available to the user through a graphical interface, known as the WorkBench, and by mouse and menu-driven operations on iconic objects. IMSE currently supports three alternative dynamic modelling paradigms and has been designed to be open to new tools and techniques in the future. The tools included at present support timed, stochastic Petri networks (PNT), queueing networks (QNET) and the process interaction view (PIT). Models are constructed by graphical editing tools, all based on a common data manipulation and graphical support system (GSS) [Uppal 1990]. In general, as all model execution will be invoked from the paradigm independent Experimenter, the user is unaware of the model solution tools.

Tools for the static modelling of systems and for the analysis of workloads are also included in the environment. The static models provide a hierarchical description of the system in terms of subsystems and offered and used services. This structural description, constructed using the *sp* [Minkowitz 1990] tool, is then available for calculating workloads and performance specifications. The workload analysis tool (WAT) allows analyses to be performed on external data to provide input for dynamic or static models. Such analyses may also be performed on the output of models for validation purposes.

A key feature of the environment is an object-oriented approach which is implemented by the Object Management System (OMS) [Titterington and Thomas 1990] at the core of IMSE. The OMS is similar to a database storing all the "objects" within the system and the links between them. Objects are derived from a common entity relationship model of the whole system and may, for example, be models, results, reports or collections of input parameter levels. The OMS provides functions that allow the tools, and therefore the user, to create and delete certain objects, group objects into directories, and to establish relational links between them.

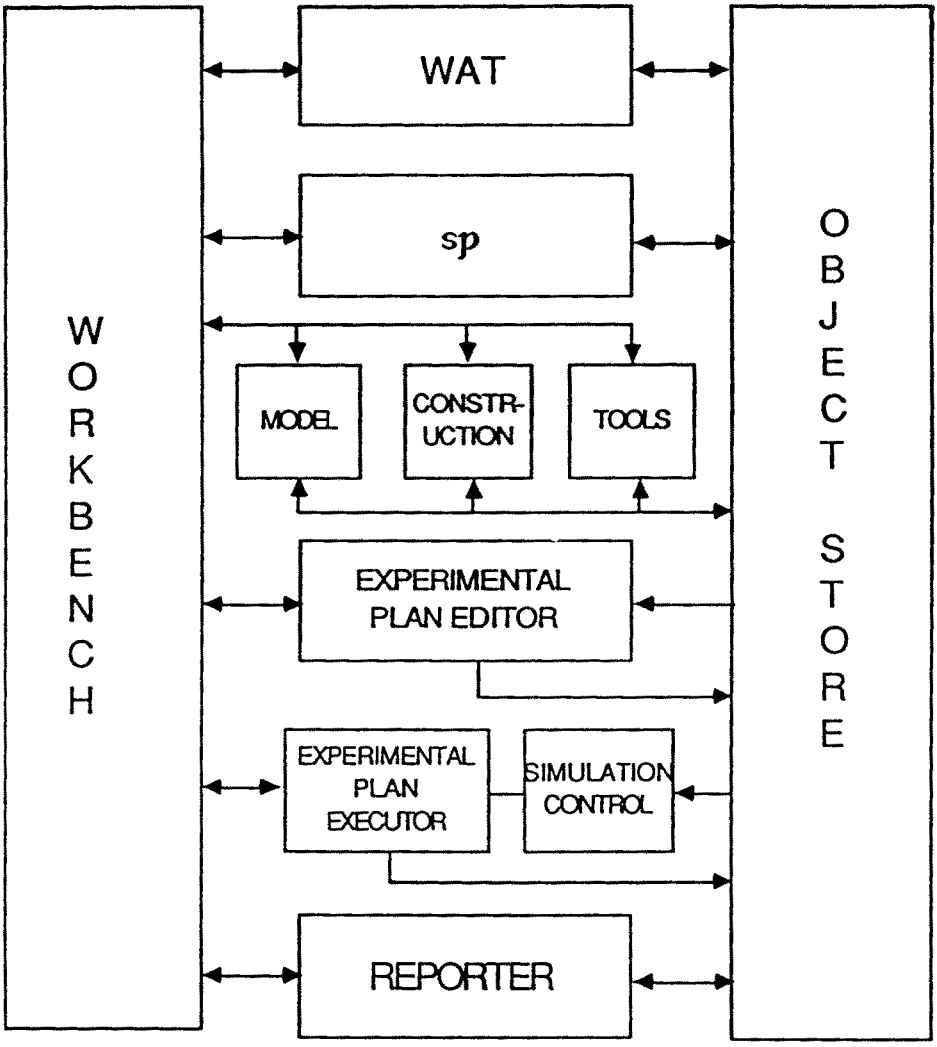


Figure 1: The IMSE Architecture

The final aspect of the IMSE environment is the tools provided to enable the user to make full use of the models created. The Experimenter, handling the models in a paradigm independent way, allows the user to concentrate on the desired conditions for model execution and the treatment of results. There is also an Animator tool, driven by simulation traces which allows the user to visualise an execution, and a Reporter tool for the creation and collation of results and reports. The overall architecture of IMSE is shown in Figure 1.

## 2.2 The Experimenter

As performance modelling has become established the process of model construction has become less important relative to the use of models and the interpretation of their results. The Experimenter reflects this change in emphasis, allowing the user to express an experiment in terms of workloads in which he is interested and results he would like to obtain [Hillston et al 1990]. The model to be used must be specified but detailed knowledge of the paradigm and solution techniques involved is often not required.

An *experiment* is deemed to be a series of related model solutions, using one or more models. The experimentation is based on clearly defined experimental plans similar to those in [Zeigler 1976], leading to an approach which is systematic and objective-driven. In other words, the set of model solutions within an experiment is designed to address a stated objective and the outcome of these solutions will be combined to form an integrated report.

The Experimenter allows different modelling paradigms to be used for model specification but also enables the resulting models to be combined within a single well-defined experiment. Previous tools such as QNAP2 and HIT allowed the analyst a choice of solution techniques from a single model specification. The use of the Experimenter within IMSE takes this approach one step further by offering the user a choice of tools for model specification as well as solution. This means that different aspects of the modelled system can

be represented, and solved, in the most suitable manner. An example of this approach can be seen in the case study presented below.

The basis of this technique is the separation of the control of model execution, directed by the Experimenter, from the realisation of model execution, carried out by the individual modelling tools. At the level of experimentation a PNT model will look the same as a QNET model. This is particularly desirable for users, such as system designers, who do not wish to be concerned with the details of the modelling. In particular it opens the possiblity of non-expert users being able to take previously constructed models and exploit them without detailed knowledge of the underlying construction.

## 2.3 Experimental Plans

The purpose of the Experimenter is to help the user to specify a "plan" of what model executions, and statistical analyses on outputs, are required, and to follow this plan automatically to produce results. Much of the work of the Experimenter in supporting multiple modelling paradigms and solution methods is new. However, some earlier work has been carried out in this field [Ören 1984; Beilner et al 1988; Umphress et al. 1989] in the support of specific tools.

The notion of an "experimental plan" is not commonly supported by modelling systems. By plan we mean a series of model solutions related by a single objective, as described above. Support is usually provided as a set of facilities for expressing multiple runs (replications, regeneration points etc.) and subsequent analysis of results. However, within IMSE an experimental plan is an object stored in the OMS in the same way as a model. Such plans can themselves be parameterised leading to the development of "generic" plans which are easily tailored to individual use. This emphasises the possibilities for the reuse of plans.

The initial development of the Experimenter design concentrated on the definition of a language in which experimental plans could be expressed. This resulted in an early prototype exploring the functionality of the tool but with only a limited textual interface. This

prototype is described in some detail in [Stevenson et al 1990]. The experimental language was the users' means of instantiating the structure of the experiment. Subsequent design work has addressed weaknesses identified by this prototype and, in particular, explored the user interface issues. As a result of this work a graphical editor for experimental plans has been developed based on the common IMSE graphics tool (GSS). Although the graphics can convey only the organisational, or schematic, information about the experiment, so that details must be entered as text, it is just such organisational information which it was found difficult to express textually. Also it is felt that the graphical approach provides the simplest interface to the Experimenter and greatest coherence with the rest of IMSE.

# 3 The Hospital Case Study

The case study presented below was started when only the textual language prototype was available. The lessons learnt formed an important input to the subsequent design and the same experimental plans were later expressed using the resulting graphical version of the Experimenter. The purpose of the study was to evaluate the whole of IMSE from practical experiences with information system engineering. Initially the evaluation was carried out with respect to the use of individual tools [Opdahl et al 1990]. As the development of IMSE progresses more of the environment will become integrated and the evaluation will be extended.

## 3.1 The System Studied

Two major criteria were identified as important in choosing the case study. Firstly, the case had to be realistic, i.e. involving a *real* performance problem on a *real* system. Secondly, the case had to be *particular* to information system engineering.

The Regionsykehuset i Trøndelag, the regional hospital of Trøndelag, agreed to cooperate in providing such a case study, centred around the hospital Patient Administration System

(PAS). System managers at the hospital identified three problem areas:

1. **Load arranging:** Interactive response times during periods of heavy-load (in the day-time) were unacceptable due to batch jobs running. Managers needed figures to convince user-departments that changing their work routines would lead to better systems response.

2. **Load balancing:** There were balancing problems in the (5 CPU-) system. System managers wanted tools to help them find better/optimal allocations of processes to the CPU's and discs in the system.

3. **Software performance evaluation:** The continuous development and installation of new applications made capacity planning impossible. Managers and application developers needed a means of estimating the workload of a system to be installed, prior to production use.

## 3.2 The Initial Objectives

The initial work, described here, concentrated on the first problem, leaving the others as reference problems for the later stages of the IMSE project. This first case study has built models of the computer system (dynamically) and existing applications (statically) and used the developing support tools to make use of these models. In particular prototype versions of the Experimenter have been used to design experiments in which the models are to run.

The problem addressed, "The Load Arranging Problem", can be stated as follows:

> The *organisation* imposes workload on the *computer system* throughout the day (24 hours). Workload consists of *interactions* and *batches*. The workload of interactions and batches follows a regular pattern throughout each day. How can batches best be *moved* within the 24 hour day in order to improve the response times of interactions?

## 3.3 The Modelling Study

It was decided that a three-level model was the most appropriate solution to this problem:

- A top-level static component model of the organisation specifying the workload on the application.
- A bottom-level static component model of the PAS application.
- A dynamic model representing the computer system.

Measurements taken from the PAS were used to validate a parameterised baseline model of the system against response times, utilisations and queue-lengths. In the top-level component of the static model a mapping could be specified to represent how work was devolved within the system to lower level components. This mapping corresponded to a particular organisation of batches throughout the day. Thus, selecting the static model component with the mapping which gave the best interactive response times for the working period would correspond to finding the best load arrangement for the hospital organisation. Once the model had been validated this reduced the problem to experimentation to find the optimal mapping in the top-level component.

Clustering analysis was carried out using the WAT, identifying the number of classes of batch jobs needed in the models. Workload parameters, service centre descriptions, and service demands were calculated from the interpreted measurement data, and from knowledge of the system in question. Models were then developed to represent the system - several QNET models to represent the dynamic behaviour and an *sp* model representing the system statically. Experimental plans to run these models were constructed. This work is described in more detail in the following section.

The case study is still in the process of being developed since a modification study is being made to investigate the consequences of moving the batch job load from peak-load intervals of the day, to low-load periods. When this work has been completed the results will be presented to the hospital's data processing managers with a recommendation for better load arranging.

# 4 Using the Experimenter

An experiment is arranged as a series of experimental frames each of which consists of a model (or a set of measurements) together with the context in which the model is to be observed in this instance. A constructed model is stored in the object store with a free parameter list, defining which parameters must be assigned values before the model can be executed, and a list of the observable outputs of the model. The context provides the specification of parameters as fixed or varied, the solution technique to be used and the results to be collected or derived.

Thus an experimental plan may be considered to be a specification of the model executions required, or rather the specification of the model executions from which results are required. If results corresponding to the given context are found to exist already within the object store these results are retrieved and no model solution is invoked. However this is transparent to the user who need not be aware that an identical model execution has already been carrried out.

The view of models supported within the Experimenter and by the underlying language is a functional one [Pooley 1989]. That is, the model is treated as a black box which responds to inputs producing outputs. Internal states of the model are only accessible via the defined output parameters.

For each model, all the free input parameters must be assigned a value before model execution can be invoked. Within the experimental plan a number of values can be associated with a model input parameter, for example values may be taken from a bounded range, an enumerated set or a probability distribution. The Experimenter will invoke a model solution for each of these values. Search strategies may also be specified by expressing dependencies between a model input value and previous inputs and outputs. Although by default every possible combination will be used when there are several varying parameters, there are facilities for constraining the search space. This gives a great deal of flexibility to the experiments which can be carried out.

The user also specifies, as appropriate, which of the possible outputs from the model are

required and the method for solving the model, and selects from the possible controls that can be applied to the model solution process. Finally the user specifies the details of how the outputs from the model executions are to be combined and interpreted to produce the results of the experiment. The format in which these results are to be displayed may also be specified, so that subsequently the results can appear on screen as a table, graph, etc., as desired.

## 4.1 Experience with the Textual Prototype

The text based prototype offered only a limited subset of the language facilities and, although this was extended for the later graphical prototype, the full functionality is only now being implemented. Some of the difficulties encountered by the case study would not have arisen if all the planned facilities had been available. Here the study served as confirmation that the features designed for the full system were appropriate.

The language which was used in the first prototype is declarative, with each frame expressed in a single block, centred around one model or set of measurements. The same model may appear in different frames - these are treated as different instances of the model, with separate assignments to the free parameters. Each frame (or block) contains the details necessary for the model's execution and observation.

### 4.1.1 An Example

The example shown in Figure 2 shows the `1CPU_1disc_QNET_experiment`, carrying out the dynamic analysis at the computer system level of the Hospital Case Study. This is a simple plan consisting of a single frame. This example illustrates many key features of the Experimenter and the underlying language.

Initially the OBJECTIVE of the experiment is stated textually. This objective is currently used only for information purposes and to underline the objective-driven approach to experimentation supported by the Experimenter. In the future it is hoped that it may be possible to attach more meaning to the objective, possibly using it to shape or check the experimental plan.

```
EXPERIMENT 1CPU_1disc_QNET_experiment

OBJECTIVE
|The purpose of this experiment is to find the average response
 time for interactive PAS customers for each of the 48 half-hour
 intervals of a 24 hour period. Batch 'response times' are also
 recorded for each interval for validation purposes.|

MODELS
\* Use the 1CPU_1disc_QNET model, which has been constructed already */

compsys = MODEL 1CPU_1disc_QNET

INPUTS
\* Declaration of the inputs - the arrival rates of each of the four job
 * classes of the compsys model. These will be provided by auxiliary experiment
 */

lambda_int lambda_ba1 lambda_ba2 lambda_ba3

OUTPUTS
\* Outputs collected are the interactive and batch class response times for
 * each of the 48 half-hour intervals being run.  For validation purposes,
 * the CPU and disc utilisation could also collected.
 */

int_resp_time ON
ba1_resp_time ON
ba2_resp_time ON
ba3_resp_time ON
cpu_util OFF
disc_util OFF

PLAN
\* Assign an auxiliary  experiment as the source of values for the free
 * input parameters.  Call the auxiliary experiment 48 times, to get the sets
 * sets of job-class arrival rates which are then fed into the compsys model.
 * Run the compsys model for each set of rates to obtain response times and
 * resource utilisations.
 */

lambda_int lambda_ba1 lambda_ba2 lambda_ba3 = EXPERIMENT SP_APPLICATION_MODEL_EXPERIMENT;

SOLVE (compsys : lambda_int lambda_ba1 lambda_ba2 lambda_ba3) BY ANALYSIS EVERY

CREATE TABLE of (int_resp_time, ba1_resp_time, ba2_resp_time, ba3_resp_time)
```

Figure 2: The 1CPU_1disc_QNET_experiment

Then all the MODELS to be used within this experiment are declared. In this case only a single model, `1CPU_1disc_QNET`, is to be used. Models within the object store together with their associated free parameter lists can be regarded as representing a class of potential models. A model used within an experiment is one instantiation of that class. To emphasise this, especially in the case when the same model "class" is used more than once within a plan, each model instantiation is given a local name within the plan - in this case `comsys`.

Each model has INPUTS which may be fixed or varied within a frame, and these are now declared. In this case the model has four inputs representing the arrival rates for interactive jobs and for the three types of batch jobs. If any of these parameters were to be fixed within this frame they would be assigned a value here.

The OUTPUTS of the model are then listed. In this experiment only response times for interactive and batch jobs are needed - these are marked as "ON". The further parameters, CPU and disc utilisations are also available as output from the model, but as they are not needed in this case they are turned "OFF". This arrangement means that the plan can easily be amended and executed again to produce a different set of outputs if the objective should change. For example, once an optimal balance of interactive and batch jobs has been established the system managers may want to know the CPU and disc utilisations resulting from this new arrangement.

The PLAN section details just how the experiment is to be carried out. There is a series of instructions for each frame within the plan:

- how values are to be assigned to the input parameters: this defines the search space in which the model is to be considered.
- the solution technique to be applied to the model: the model solution tool to be used is inherent in the model construction, for example QNET for a queueing network model. However this tool itself may offer a choice of model solution techniques.

```
EXPERIMENT SP_application_model_experiment

OBJECTIVE
|The purpose of this subexperiment is to devolve the workload put on the
 computer system model by the hospital use of the PAS application during
 a half-hour interval.|

MODELS
\* Use the SP_application_model, which has been constructed already */

appl = MODEL SP_application_model

INPUTS
\* There is one input to this model - the number of half-hour periods */

hh

OUTPUTS
\* The outputs are the arrival rates for each of the interative and batch jobs
*/

lambda_int ON
lambda_ba1 ON
lambda_ba2 ON
lambda_ba3 ON
PLAN
\* Range hh over the complete 24 hour period and for each half hour collect
 * the arrival rates of each job class.
 */


hh RANGE from 0 to 47

SOLVE (appl : hh) BY ANALYSIS

CREATE STREAM of (lambda_int lambda_ba1 lambda_ba2 lambda_ba3)
```

Figure 3: The SP_application_model_experiment

- the results to be constructed from the model outputs.

In this example the inputs are to be taken from another EXPERIMENT. This is termed an auxiliary experiment and is an experiment carried out in order to produce results which are then used as input to a model. In this case the static model representing the devolution of workload from the PAS application to the computer system is used to generate the interarrival rates of the jobs in the computer system. This auxiliary experiment is the experiment SP_application_model_experiment, shown in Figure 3.

In the 1CPU_1disc_QNET experiment the model comsys is to be solved by ANALYSIS. Similarly in SP_application_model_experiment the model appl, an instantiation of SP_application_model, is also to be solved by ANALYSIS.

From the auxiliary experiment there is no requirement for a constructed results object, such as a table or graph, since outputs are collected to be used as input in the `1CPU_1disc_QNET_experiment`. However, in the 'parent' experiment a TABLE is to be constructed showing how all the response times vary. The values generated will in fact correspond to the different half hour periods throughout the day, since this was the basis for the adjustment of the input parameters.

### 4.1.2 The Evaluation

Much of the early evaluation effort concentrated on the syntax and structure of the language. In the later graphical prototype many of these points were no longer directly applicable. However it was thought that, since the language was at the core of the Experimenter, it would always have a strong influence on the interface and the "flavour" of the tool, even though it might not be explicitly visible to the user. Therefore subsequent design work paid careful attention to these criticisms.

The facility for auxiliary experiments, using one experiment (the auxiliary) to provide input values for another (the main experiment), was found to be very useful. For example, in the hospital case study different models were used to capture different aspects of the system yet these could be combined within a single experiment. This is seen as one of the greatest strengths of the Experimenter, allowing different views of a system to be combined even though developed in separate models, and possibly using different paradigms.

As explained earlier, the conceptual view taken of an experimental plan is that it is objective-driven and made up of one or more experimental frames. However the textual prototype revealed some indecision on the part of the designers as to how much of this conceptual structure should be apparent to the users. It was felt that the syntax used in the prototype represented a compromise which was confusing to the users. They were aware that some definite structure was being imposed on the model executions but uncertain about its nature.

Use of the prototype pointed out that the handling of parameters was not always consis-

tent or appropriate. A strong distinction had been made between the declaration of the parameters involved in an experiment and how they were to vary during the experiment. This was found to be cumbersome. Also, experience of writing the experimental plans used in the case study suggested that it might be useful to have some means of grouping related parameters. For example, input parameters were declared, assigned values and possibly included in an analysis method, the same sequence of variable names occuring several times. It was felt that the amount of editing and the likelihood of errors would be reduced if some grouping construct was available.

During the course of the case study it would have been useful to be able to use the Experimenter to compare different models of the same system under the same conditions. Although this was possible using the textual interface the users found it difficult to express. This type of comparison is clearly important, especially with respect to the validation of models. It was recognised that the facility needed to be made more accessible to the users.

## 4.2 Experience with the Graphical Version

The graphical prototype was not merely a graphical representation of the experimental language. Based on the responses to the first prototype and a better understanding of the requirements on the Experimenter, the underlying language was modified and further developed. Thus the graphical prototype offered greater functionality and a slightly different view of experimental plans.

### 4.2.1 The Example Revisited

The plan is now represented as a series of nodes, each of which has associated attributes representing the details of the plan. "Opening" a node produces a form which prompts the user to supply the information relevant to that aspect of the plan. The experiments described in the previous section are shown in the graphical representation in Figures 4 and 5. In this representation the frame is represented as a collection of nodes. The central node represents the model or set of measurements under consideration. Opening

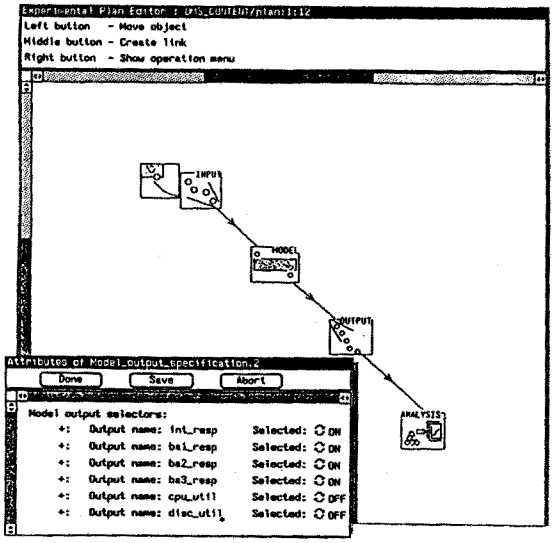


Figure 4: The graphical representation of the main experiment

this node the user is offered a form prompting for the name of the model (measurements), the solution technique to be used and any control parameters defining how the solution is to be reached.

Linked to this node are input and output nodes listing the input and output parameters for the model concerned. As previously, the user is required to provide values to be assigned to each input parameters. This may be a single value making this parameter fixed within this frame or a definition of how the parameter is to vary. When an auxiliary experiment is to be used as a source of values it is denoted by a subnode as shown in Figure 4.

The form associated with the output node lists the possible outputs of the model. The user selects which of these are required by means of a simple on/off toggle switch. Each frame is connected, via the output node, to one or more analysis node. This represents the results to be produced by the experiment, collected or derived from the model parameters. The form associated with this node offers the user a list of the input and output parameters

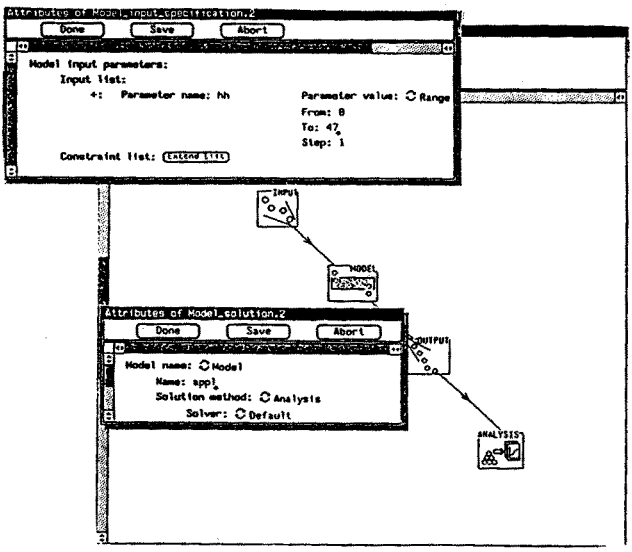


Figure 5: The graphical representation of the auxiliary experiment

of the attached frame, and a variety of possible results objects.

### 4.2.2 The Re-evaluation

The graphical prototype was very well received. It was much easier and much quicker to use: since the user has only to select and position nodes and enter values into forms an experimental plan can be built in a matter of minutes. Provided with a better user interface, it was easier for the users to concentrate on the functionality of the tool. In some cases it became much clearer how a plan can be constructed to meet a particular objective. For example, in the case of comparing two models of the same system, two frames (model, input and output nodes) can be built and linked into the same analysis method.

This new style of interface immediately removed many of the problems which had been encountered with the first prototype. The language was still at the core of the Experimenter but became embedded in the forms used to enter information, and so far less obtrusive

for the user. It had been noted that the users had felt uncomfortable with the lack of organisational aspects of the language, such as delimiters and keywords. Hence care was taken to ensure that the forms were well annotated and clearly structured. Since the use of graphics left the user free to construct the plan in whatever order was convenient another previous problem was removed.

As can be seen in Figures 4 and 5, in the graphical representation of an experimental plan the experimental frames are clearly recognisable as a collection of an input, output and model node but this structuring is not made explicit to the user. As this is a natural grouping of these pieces of information the user need not be aware of the underlying theoretical structure on which it is based.

There were other usability issues which were not so simply addressed by the change of interface style. The manner of handling parameters was reconsidered in some detail. Since the separation of the declaration and assignment to parameters seemed unwieldy during the case study it was decided to incorporate the declaration implicitly within the specification of how the parameter is to vary. However problems still arose in the graphical prototype with respect to the handling of parameters. When the user entered values in the input forms these were automatically type-checked against the type specified for the parameter at the time of model construction. If the types did not match, the value was rejected. It was felt that it would be more helpful if the type expected was displayed to the user. In the current implementation the form associated with the input node displays not only the parameter's name but also a default value for the type and any annotating text which was supplied at the time of model construction. This is felt to be particularly useful when the person using the model is not the person who constructed the model originally.

Although the grouping of parameters was considered, it was felt that this was no longer a necessary feature after the introduction of the graphical interface. The parameters involved are derived automatically from the model when it is constructed and are presented to the user in the experimental plan once the model to be used has been specified. In this way the user does not need to enter the names of parameters, merely to supply their

values. These values are automatically type-checked to reduce the likelihood of error. All input parameters and collected output parameters are available for selection by the user in the form associated with the analysis node. "Cut and paste" facilities are available in the graphical tool used to construct the interface so that parameter's assignments can easily be copied if necessary.

In general, making experiments and the use of models more accessible to people other than modelling experts was considered in much more detail with the graphical prototype. The textual prototype had been unsuitable for naïve users but the graphical version opened up this possibility and it was felt that this direction of development should be accentuated. This resulted in an emphasis of simplicity and ease of use for the interface. It also emphasised the requirement for previously prepared plans to be used and tailored by a naïve user.

Another useful feature, the ability to parameterise experimental plans, was also considered. Parameterisation of plans in terms of leaving undefined some of the input parameters to the models involved has been incorporated as part of the functionality of the Experimenter. This was originally seen as being particularly useful for auxiliary experiments where it might be necessary for the auxiliary and main experiments to share a parameter value. However there is now a requirement for leaving undefined more structural properties of the experimental plan, such as the model to be used. The first development does not include this feature, but it should be implemented to some extent in later work.

# 5 Further Developments and Future Work

The case study found that the Experimenter was a useful tool with a strong impact on the modelling process. In a study involving hundreds of runs and several models, it would be very easy to lose track of what was being done and of how models related to one another. Expressing experimental plans in the formal framework of the Experimenter, and keeping these plans updated, made the study easier to organise and conceptually cleaner.

Particularly in the later stages, the evaluation was carried out from the point of view of users who want to be aware of as little as possible of the underlying tools. It was felt that the graphical prototype went some way towards addressing the needs of these users and showed the potential for much more work in this area. The continued interaction between the tool designers and the evaluators is expected to encourage this work.

The production version of the Experimenter is currently being implemented and fully integrated within IMSE. It is hoped that its use with the fully developed modelling tools will highlight the potential it provides. Future work will be looking at the ways in which this functionality can be enhanced, both by making existing facilities more accessible, especially to non-expert users, and by adding new facilities.

The Experimenter raises the use of models to a higher level of abstraction and therefore is ideal for the user who is unfamiliar with the details of model construction. Previously these users were restricted in what they could achieve with models by their own lack of knowledge of how the model was constructed. However, the view of a model as a black box with clearly defined interfaces switches the emphasis from the model itself and the details of its construction to the use of the model and the interpretation of its results. With such a model-independent interface much more flexibility can be made available to naïve users.

Enhancing the use of models in this abstract way emphasises the properties of the models independently of the paradigm in which they are expressed. Separating the use of models from the construction of models without restricting that use is an important step towards a close working relationship between system design and modelling.

## Acknowledgements

The authors would like to thank Neil Stevenson of Edinburgh University and Vidar Vetland and Arne Sølvberg of SINTEF for their invaluable contributions to this work and all the other IMSE teams for their help.

# References

N. Abu El Ata (ed)., *Modelling Techniques and Tools for Performance Analysis '85*, North Holland, 1986.

H. Beilner, J. Mäter, and N. Weißenberg., "Towards a Performance Modelling Environment: News on HIT", in Puigjaner 1988.

J. Hillston, R. Pooley, and N. Stevenson., "An Experimentation Facility Within the Integrated Modelling Support Environment" in Proceeding of the UKSC Conference on Computer Simulation, Brighton, September 1990.

P.H. Hughes and D. Potier., "The Integrated Modelling Support Environment" in *Esprit Information Processing Systems: Results and Progress of Esprit Projects 1989*, CEC DGXIII, 1989.

C. Minkowitz., "The Structure and Performance Specification Tool Design Document", IMSE Document D4.1-1. STC Technology Ltd, Newcastle-under-Lyme, England, 1990.

A.L. Opdahl, V. Vetland and A. Sølvberg., "Information Systems Engineering: Evaluation of the IMSE", IMSE Document D6.6-1. SINTEF, University of Trondheim, Norway, 1990.

T.I. Ören., "Gest: A Modelling and Simulation Language Based on System Theoretic Concepts", in *Simulation and Model-Based Methodologies: An Integrative View*", T.I. Ören, B.P. Zeigler, and M.S. Elzas (Eds), Springer-Verlag 1984, pp 281-335.

R.J. Pooley., "Hierarchical Simulation and System Description", in *Proceedings of the 3rd European Simulation Congress 1989*, Edinburgh, 1989.

R.J. Pooley., "An Experimenter Tool for an Integrated Modelling Support Environment - its Rôle and Design" in *Software Engineering Journal*, Vol 4 No 4, pp 163-170, May 1989.

D. Potier (ed)., *Modelling Techniques and Tools for Performance Analysis*, North Holland, 1985.

R. Puigjaner (ed)., *Proc. Fourth International Conference on Modelling Techniques and Tools for Computer Evaluation*, Plenum Publishing, 1988.

N. Stevenson, J. Hillston and R. Pooley., "A Tool for Conducting Modelling Studies" in *Proceedings of 4th European Simulation Multiconference*, Nuremburg, 1990.

G. Titterington and A. Thomas., "The IMSE Object Management System", IMSE Document D2.1-5, STC Technology Ltd, Newcastle-under-Lyme, England, 1990.

D.A. Umphress, U.W.Pooch, and M. Tanik., "Fast Prototyping of a Goal-Oriented Simulation Environment System" in *The Computer Journal* Volume 32, No 6, (December) 1989.

C. Uppal., "The Design of the IMSE Structured Data Manipulation Facility", IMSE Document D2.2-1. STC Technology Ltd, Newcastle-under-Lyme, England, 1990.

B.P. Zeigler., *Multifacetted Modelling and Discrete Event Simulation*, Academic Press, 1984.

B.P. Zeigler., *Theory of Modelling and Simulation*, Krieger, 1976.