

# **Modelling and solving environments for mathematical programming (MP): a status review and new directions**

B Dominguez-Ballesteros, G Mitra<sup>\*</sup>, C Lucas and N-S Koutsoukis.

Brunel University, Department of Mathematical Sciences, Uxbridge, Middlesex, UB8 3PH

Languages and computing environments that support Mathematical Programming (MP) modelling continue to develop and evolve. In this paper *(i)* we address declarative and procedural features of modelling languages. *(ii)* We assess developments which couple the modelling and the solving processes, typically column generation, the branch & price approach to Integer Programming (IP), and the sampling of scenarios within Stochastic Programming (SP). *(iii)* We consider how data modelling and symbolic modelling naturally come together and are used within the information value chain. *(iv)* Finally, we investigate the features of new tools, which support prototyping of optimisation based decision support (DS) applications.

**Keywords:** Linear Programming, Planning, Algebraic Modelling Languages, Optimisation, Decision Support.

\* e-mail address: [Gautam.Mitra@brunel.ac.uk](mailto:Gautam.Mitra@brunel.ac.uk)

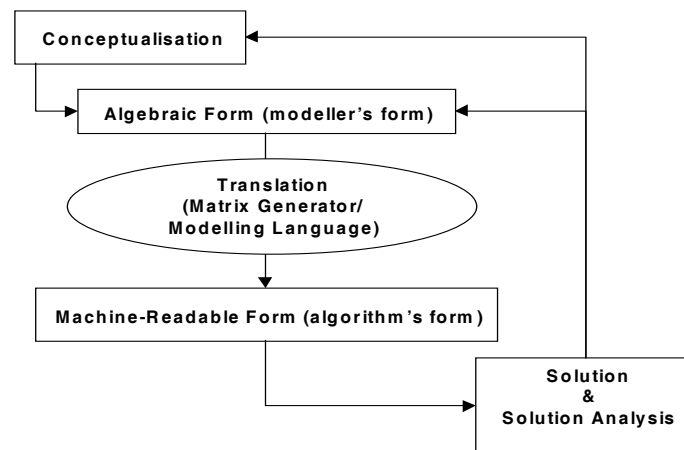
---

## Introduction & Background

There is a re-emergence of interest in Mathematical Programming (MP) models for optimum Decision Support (DS) applications. To construct such DS applications it is often necessary to undertake feasibility studies, and construct and investigate optimisation models. This paper focuses on the tools and software environments, which support such investigations.

### *The MP Modelling Process*

For an understanding of how MP is applied to industry it is essential to recognise the different stages in the process of model formulation, solution and results investigations.



*Figure 1 Stages in Formulating LP Models (prior to the 90s)*

The different stages of the overall process as set out in Figure 1 are explained in the following way.

**Conceptualisation:** Initially the problem owner analyses the real world problem. The available information is collected and used to study the 'real' issues; these are to establish the content and the relevance of the given problem, and what needs to be optimised. At this stage it is not necessary to think about the mathematical formulation of the problem in the form of an algebraic model. Instead it is necessary to consider the physical problem and the essential aspects which must be taken into account for decision making.

**Algebraic form:** Subsequently the modeller develops a mathematical formulation of the problem. In order to model a given problem as a MP it is necessary to specify an

objective function to be maximised (or minimised) as well as the algebraic constraints in a suitable algorithmic form. This leads to *constrained optimisation* problems. This mathematical representation of the real problem is generally divided into blocks corresponding to the model subscripts and dimensions, the model coefficients, the model decision variables, the constraint relations, and the function to maximise (or minimise). Lucas and Mitra <sup>1</sup> propose that we move from conceptualisation to an algebraic specification by following a series of logical steps, which are:

Step 1. Define the subscripts and their ranges (sets and dimensions).

Step 2. Define the data tables in terms of the sets and the dimensions defined in Step 1.

Step 3. Define the model decision variables in terms of the sets and the subscripts defined in Step 1.

Step 4. Identify the objective and specify it as an algebraic expression using the decision variables defined in Step 3.

Step 5. Interpret the restrictions and specify these as algebraic relationships in row-wise fashion using the decision variables defined in Step 3. The relations are labelled using the subscripts and sets of Step 1.

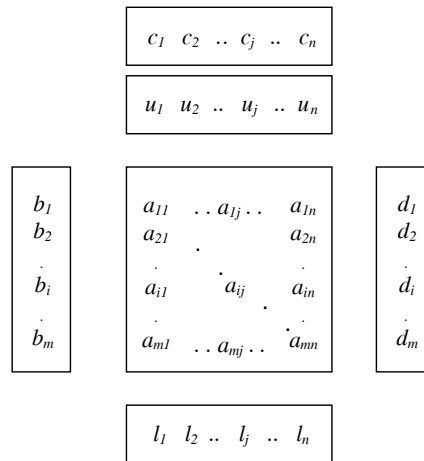
The algebraic specification is typically cast into a machine-readable form with the use of a modelling tool. With such a tool modellers describe mathematical programs in a readable and symbolic form known as the “modeller’s form”,<sup>2</sup>. An example based on Linear Programming (LP) is shown in Figure 2.

Find $x \in \mathfrak{R}^n$ :	
Minimise (or maximise)	$\sum_{j \in J} c_j x_j$
Subject to	$b_i \leq \sum_{j \in J} a_{ij} x_j \leq d_i$ Linear Form Constraints
	$l_j \leq x_j \leq u_j$ Bounds Constraints
Where	$c_j, b_i, d_i, l_j, u_j \in \mathfrak{R},$
	$a_{ij} \in \mathfrak{R}, (a_{ij}) \equiv A \in \mathfrak{R}^{m \times n}$ Coefficients Matrix

**Figure 2 General LP Algebraic Formulation (Modeller’s Form)**

The model is presented in an algebraic form using the  $\Sigma$ -notation. This notation is used to specify a linear form corresponding to the objective function as well as one or more linear form restrictions and bound restrictions which must be satisfied. The resulting LP is an abstract representation, which must be analysed and subsequently solved in order to find an optimum solution to the given decision problem.

**Machine readable form:** LP can be viewed from a different perspective: for an algorithm to process this model, a numeric representation of individual row and column data must be provided. Thus, in contrast to the “modeller’s form”, this “algorithm’s form”<sup>2</sup> no longer maintains the algebraic representation, but represents the model as arrays of numbers.



**Figure 3 Matrices in a General LP (Algorithm's Form)**

In Figure 3 all the coefficients of the constraints matrix, and those of the objective function, lower and upper bounds of the variables, right hand side and left hand side constraint values are shown explicitly. These values can then be read by a solver, which processes it to compute the optimum solution.

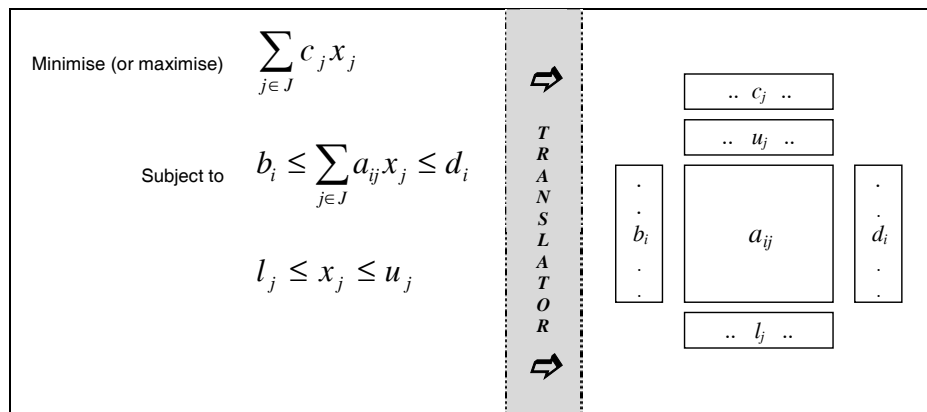
<b>Machine Readable Form</b>
Standard (MPS)
Internal (AIMMS, AMPL, MPL)

**Table 1 Machine Readable Forms**

The standard machine-readable form of LP models is MPS,<sup>3</sup> where the matrix definition is in a sparse column oriented format. The MPS form of an LP is the

analogue of the assembly language form of an algorithm.<sup>4</sup> It consists of sections introduced by header lines for the different components in a model (name, rows, columns, right-hand-side constraints, and bounds). However some modelling languages, such as AIMMS, AMPL, and MPL use proprietary formats as an alternative to the MPS format. These specific formats can be created faster than the MPS format, and can be generalised to other types of models, like quadratic programming models. It is also possible to extend the use of these specific formats to support other modelling paradigms such as stochastic programming (see SMPS),<sup>5</sup> and constraint logic programming.

**Translation:** To progress from the modeller’s form to the algorithm’s form requires a transformation procedure called Translation.<sup>2</sup> The translator in MP languages plays the same role as that of a compiler in computer languages.<sup>6</sup> This step involves the analysis of the algebraic syntax in order to explicitly generate the coefficients of the constraint matrix. The translation process is represented in Figure 4.



**Figure 4 Translation from Modeller's Form into Algorithm's Form**

As in a programming language compiler the algebraic modelling language statements are first parsed and then a code generation process is invoked, which leads to the creation of the matrix (model) coefficients. Over the last two decades *MP Modelling Languages* have gained considerable acceptance as a natural tool for specifying MP problems in a form, which is easy to understand, and communicate between analysts. In this approach the MP modelling languages are used as a ‘translator’,<sup>2,7</sup> which take the ‘declarative’ algebraic statements specifying the model, interpret it, and generate executable code. Consequently, a modelling language is a computational tool which connects the modeller’s form and the algorithm’s form. It gives the modeller the option to represent the model in a user friendly form and allows him to focus on the

problem at hand rather than writing a translation program. It is useful to contrast this tool against a matrix generator for LP whose sole purpose is to create the matrix coefficients for a particular model. This is one of the main advantages of using modelling languages, as they are model-oriented, while matrix generators are solver-oriented.<sup>8</sup> Modelling languages and matrix generators differ fundamentally in how they divide this task of translation between human and machine.

The form in which the model and the data are presented determines which tool is used. Specific differences between modelling languages and matrix generators include the (in) dependence of the algorithm's form; in the matrix generator the model structure is seen as a matrix data, whereas in the modelling language the model is represented as a set of relations (equations). For more details see <sup>2</sup>.

Many LP and IP problems cannot be stated immediately in a linear form; typically 'column-oriented' modelling, that is, 'extreme points' representation allows a modeller to capture some complex set of rules or requirements of a given problem domain - see Gilmore & Gomory cutting stock,<sup>9</sup> crew scheduling,<sup>10</sup> and U.S. Coast Guard cutter scheduling.<sup>11</sup> Hence declarative languages have been extended to include procedural features so that these classes of models can be captured and represented.

**Solution:** The solver engine encapsulates a solution algorithm, which is capable of understanding and processing the matrix data in the algorithm's form and produce results which are optimal or inform that the model is not solvable. Some typical solution algorithms for LP are the Simplex Method,<sup>12</sup> and the Interior Point Method (IPM) algorithms.<sup>13</sup> Solution methods continue to be focus of ongoing research.<sup>14,15,16</sup> From a conceptual viewpoint, the processing by a solver may be considered to be an inference step. From this perspective we can call the solver the *decision engine* within the entire system. The use of one or another specific algorithm depends on the particular problem under consideration. Our predominant interests lie within LP and IP solvers, such as CPLEX,<sup>17</sup> FortMP,<sup>18</sup> XPRESS-MP.<sup>19</sup> However, in recent times there has been growing interest in non-linear solvers, and considerable interest in applications of logic-based inferencing procedures. Traditionally, for the problem owner the solver is a black box with limited control over the solution process. However, this inaccessibility is being reduced, as the requirements from different applications grow. The modeller can also interact closely with the solver by using procedural scripts that are part of the model.

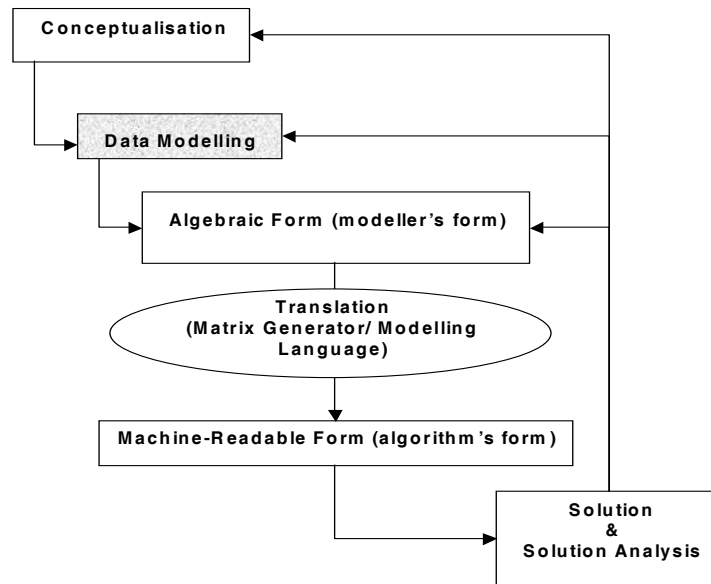
**Solution analysis:** After processing by the solver the solution (result) is typically stored in a solution file, and would include the objective value (or the status of the solution like ‘unbounded’ or ‘infeasible’), the values of the decision variables, and other related solution values, such as the values of logical variables associated with each constraint, shadow prices, and reduced costs. These items are generally considered to be decision data. The decision-maker or analyst studies these decision data as inferences made using the model. Alternatively, the decision-maker may experiment with different parameters of the model or return to a previous stage to improve the model formulation.

### ***Integrating within the Information System Architecture***

To make optimisation widely available to the user community it is necessary to provide computer support for model formulation and model solution. In the 70s and 80s there was a major development of optimisation algorithms for efficient solving of large-scale problems. However the development of tools for formulating and investigating MP problems was very limited. It is only since the early 90s that MP modelling and analysis tools have gone through a rapid evolution and growth. In this way MP modelling tools have been able to keep up with the progressive acceptance of optimisation. MP models are used to find optimal solutions to a particular problem, as they are deployed to address decision problems, which usually draw upon large sets of carefully collected and accumulated data sets. MP models are therefore considered as critical components of an organisation’s analytical Information Technology (IT) systems. MP models are also used to measure and analyse performance indicators (business measures) such as cost, profit, quality, and time.<sup>20</sup> This practice has been addressed by bringing together MP modelling tools and database technology. As a result, the use of database technology has become a main activity for MP data storage and data analysis. From the data analysis side, these modelling environments connected to database systems are gaining increasing acceptance as decision support tools, since the reports and solution analysis results are put back in the same analytic database. These are referred to as *decision data*.<sup>21</sup> From the data storage viewpoint a natural step of Data Modelling is added within the process of formulating MP models:

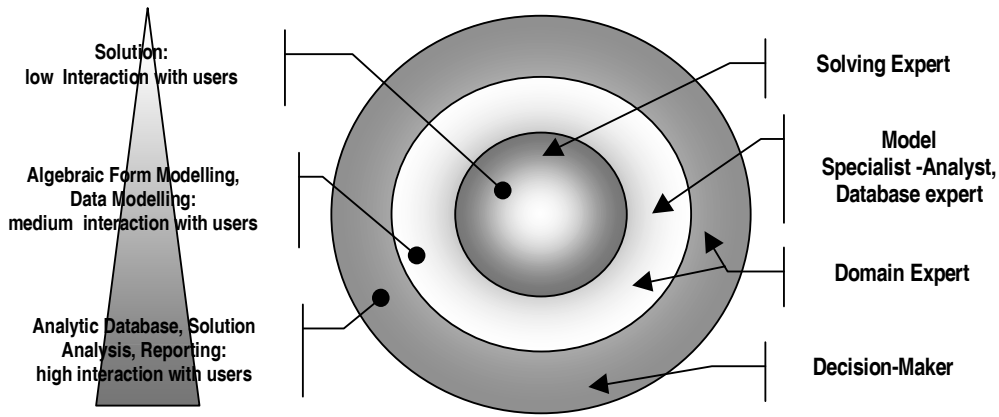
**Data modelling:** The modeller extracts, classifies, and stores information related to the decision problem, in order to use it in subsequent stages such as the (MP) model formulation. The items of information are analysed and subdivided into ‘categories’,

which in turn provide the sets and indices of the algebraic model. For this data evaluation there is a need to categorise and extract the available data related to the problem.<sup>22</sup>



*Figure 5 A Recent View of the Process in Formulating LP Models*

**Constituents:** In the different stages of the optimisation process, various constituents undertake different roles and interact with each other. These can be seen in Figure 6.<sup>23,24,25</sup> The *Technical Experts* (*modelling experts, database experts, solver experts, and domain experts*, depending on their expertise) have access to all system components including the underlying models, databases, and can change the solver strategy. The different Technical Experts collaborate to create a domain-specific application by integrating the different tools for the different stages of the process. The *Decision Maker(s)* are the constituents that utilise the (application) system. Typically the Decision Makers have access to the data and solution analysis routines available in the database, and use the underlying models and data as a black box. These are only interested in the solution values, which are expressed in terms of decisions. Hereafter, they analyse and report the results that are in the analytic database.



*Figure 6 Constituents and Level of End-User Interaction*

Accordingly, there is a need to have integrated optimisation tools for all the constituents involved. These tools must be capable of providing a set of features that ideally allow the different experts and problem owner(s) to complete the life cycle of the LP based optimisation application.

The rest of this paper is organised as follows. We introduce relevant features and extensions within the MP modelling languages. The close coupling between modelling and solution tools is presented later on. Soon after we set out a sequence of steps which capture the model and results analysis. We consider the positioning of MP modelling within the information systems architecture, and finally we discuss new directions for the development of modelling systems, and draw some conclusions for further research.

## Modelling Features and Extensions

In order to model a given problem as a mathematical program it becomes necessary to specify an objective function as well as the constraints in a suitable algebraic form. This can be achieved by using MP modelling languages, which are designed to express the modeller's form of the problem in an accessible and comprehensible format for the user, while hiding the unfriendly mathematical notation.

Kuip<sup>4</sup> considers languages that support LP programs and languages for programming algorithms from an alternative viewpoint as shown in Table 2.

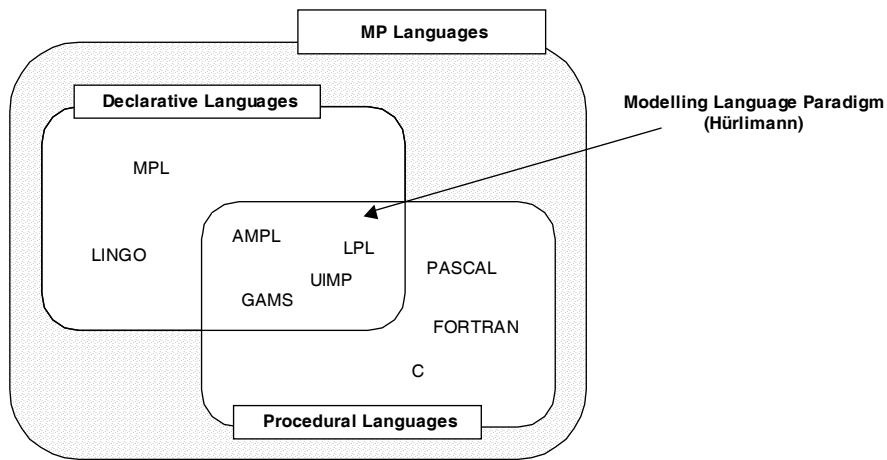
Language for Linear Programs	Language for Programming Algorithms	Basis of similarity
Algorithm's form	Machine code	Executable representation
MPS form	Assembly	The use of mnemonics
Matrix generator	Procedural	Grouping of mnemonics
Algebraic	Declarative	Resemblance to mathematical notation

*Table 2 Classes of Modelling and Programming Languages*

According to this comparison we consider algebraic languages as declarative languages. This is the case of many MP languages such as CAMPS,<sup>1</sup> LINGO,<sup>27</sup> MPL,<sup>26</sup> that restrict their syntax mainly to the algebraic syntax. However, other modelling languages like AIMMS,<sup>30</sup> AMPL,<sup>29</sup> GAMS<sup>31</sup>, UIMP,<sup>28</sup> or the modelling facility of XPRESS-MP,<sup>19</sup> include procedural language extensions such as If-Then-Else statements and other capabilities, such as the use of procedures or routines, similar to those found in a programming language. For example, UIMP incorporates explicitly in the syntax a hierarchical table structure often arising in real-life MP models.<sup>28</sup> These additional characteristics enrich the functionality of the algebraic notation. They provide the ability of customising different parts of the solution process such as column generation, which is exploited in applications such as the cutting stock or crew scheduling problems. Hürlimann<sup>32</sup> uses the term *modelling language* to describe a new paradigm of programming, namely MP modelling languages which combine declarative and algorithmic knowledge.

Taking into account these fundamental language differences, it can be stated that languages used for mathematical modelling belong to two distinct classes, namely

*Declarative Languages* and *Procedural Languages*. In practice, however, some of the languages have features of both classes, as we illustrate in Figure 7.



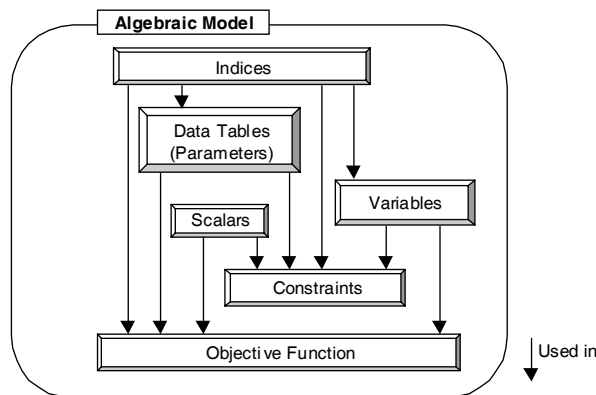
**Figure 7** Modelling Languages Classification

Procedural languages such as FORTRAN and C are not categorically MP modelling languages; nevertheless we consider them here because they are frequently used to construct matrix generators and solution analysers acting as low level alternatives to MP modelling tools.

**Declarative Features**

Syntax of the Linear Form

The algebraic models have in general, as main components, Indices, Parameters (data), Decision variables, Constraints, and an Objective function. The relationships between these components are illustrated in Figure 8.



**Figure 8** Main Components in an Algebraic Model

That is, the indices can be used in the definition of the data tables, the variables, in the constraints, and in the objective function. In the same way, data tables, variables, and scalars are used to compose the constraints and the objective function.

An aspect of the syntax of the linear form is the representation of the model, which may be column-oriented, row-oriented, or block-wise-oriented. The choice of the type of representation depends mainly on the type of application, the background of the modeller, and the functionality of the modelling language.<sup>4</sup> In particular AMPL caters for all three representations. Typical examples of column-oriented representation problems are cutting stock problems<sup>9</sup> and scheduling problems.<sup>10,11</sup> One specific case of column-wise representation is the network structure.

One of the most persuasive characteristics of the syntax in the algebraic modelling languages is the use of *subscripted terms (variables and parameters)* and *index supporting functions* to represent and manipulate families of objects.<sup>7</sup> Using subscripted terms a family of variables is written by varying the subindices through their range without having to declare one by one all the variables that express the same concept. Similarly this is done for model coefficients. Index supporting functions, following the same principle, are used to specify the objective function and the constraints. The objective and the constraints characterise the algebraic form of an LP. It is in the constraints section that we exploit algebraic terminology by using not only summations over sets of indices, but also arithmetic expressions. The constraints section is usually considered the most important part of the entire model description, as it forms the basis for the LP matrix to be constructed.<sup>4</sup>

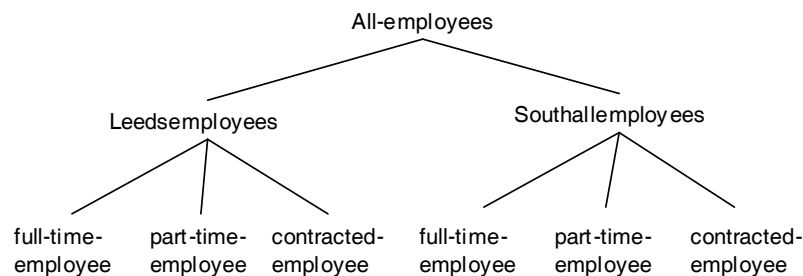
## Sets and Indices

Algebraic MP languages allow the use of *Indices* and *Sets*. These are one of the most powerful features of algebra when dealing with large-scale problems, as indices and sets facilitate the classification of elements that have a similar nature.

An *index* is a number, an identifier, a string of characters or an expression that has a number or string as a value. An *index-set* or simply a set, is a countable and finite collection of indices or tuple of indices. Some modelling languages consider a set as an ordered collection of elements. This assumption does not provoke a loss of generality, as an unordered collection can always be represented by an ordered one.<sup>33</sup>

Indices are typically used to represent different types of sets. These can be, among others, *Unordered* or *Ordered sets*, *Ordered tuples*, *Compound sets (sets of sets)*, and *Hierarchical sets*. Compound sets (sets of sets), which are sets of pairs, triples, or longer tuples can be combined with the union, intersection, difference, and general operations allowed for simple sets. Compound sets can be considered as a special case of hierarchical sets, where the arity is the same for all tuples that compose the set. In hierarchical sets the elements can be tuples of different arity. This type of set supports hierarchical structures with parent-child relationships. In practice the number of levels supported is low. They can be represented by a tree structure (Figure 9) called the *index-tree*.<sup>34</sup> Currently LPL,<sup>34</sup> UIMP,<sup>28</sup> and AMPL<sup>29</sup> include hierarchical sets, although they do not exploit them sufficiently.<sup>4</sup>

e.g., `factoryemployees = {Southallemployees, Leedsemployees},`  
`employeetype = {full-time-employee, part-time-employee, contracted-employee}`



**Figure 9 Index-Tree of a Hierarchical Set**

New types of index sets can be created by using the *Standard set operations*, which are Union, Intersection, Difference, Cartesian Product, Selection, and Join.

Another extension of the algebraic notation, due to the indexing capability, is to allow recursive definition of intermediate expressions, which is useful in some types of problems such as dynamic models. For example:

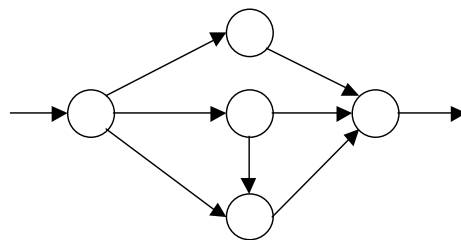
`Inventory [p, f, timeperiod] =`  
`ProdQty + Inventory [p, f, timeperiod:=timeperiod-1] - SUM(d: SendQty)`

There are additional reasons why indices are important in algebraic modelling languages. A model can be viewed as a model class, with a generic structure, that is instantiated by changing the index-sets. The model remains the same whereas the range of the indices changes for each instance. This feature allows the modeller to

use the model to investigate different real life situations, to communicate the model to other modellers, to verify the model more easily, and to concentrate on the structure of the model.

## Network Structure

The network structure arises in models for many real life problems, such as transportation, assignment, and transshipment problems. Usually this structure is represented as a graph, where the constraints are associated with nodes, and flows along the arcs relate to the decision variables.



**Figure 10 Graph Representation of a Network**

The formulation of this type of problems specifies the amount of flow on the arcs (variables) while maintaining flow balance between nodes (constraints). The flow balance equation is compactly written algebraically, but is difficult to express in a declarative form. A typical flow balance equation takes the form:

$$\sum_{(k,j) \in A} x_{(k,j)} = \sum_{(i,k) \in A} x_{(i,k)} + d_k \forall k \in N$$

where  $i, k, j$  are node indices,  
 $(k, j)$  define an element of the arc set  
 $N = \{\text{nodes}\}, A = \{\text{arcs}\}, d_k = \text{demand}_k$

The first summation shown in the equation above reads as “sum over all nodes that have an arc joined to node  $k$ ”. This is difficult to introduce as a summation command within an algebraic language, unless a mechanism is found to represent the node and arc structure of the underlying network.

AMPL and AIMMS address this issue in the language structure whereby the objective and constraints are not specified by a conventional algebraic statement. These two modelling languages introduce two special groups of declaration, *arc* and *node*, that

substitute *var* and *subject to* respectively in an algebraic formulation. This extension in the language helps formulating these models, since people see network flow problems as an explicit group of nodes, arcs, from which flow variables and balance constraints can be extracted, and not the opposite as in the conventional algebraic approach.<sup>35</sup> A comparable approach was developed by Zenios<sup>36</sup> using a pre-processor between the algebraic modelling language and the network optimiser to capture such structure in the model.

### Unit Consistency

Bradley & Clemence<sup>37</sup> were first to highlight the importance of introducing unit consistency as part of their proposed modelling language, EML. In this MP language they assign a type to each variable, coefficient, constant, function, constraint, input and output of the model that consists of its “concepts”, “quantities” and “units of measurement” with optional scale factors.

Bisschop<sup>38</sup> points out that one of the most frequent errors in modelling is the inconsistency of measurement units (unit errors). Modelling languages that support unit definitions for each model component not only automate the process of unit conversion and scaling, but also enhance readability of the model, and reduce the scope of syntax errors.<sup>34</sup>

#### DATA

```
RawMatCost          := 3.00;          [$ / lb]
RawMatYield[perfume] := (3, 4);       [oz / lb]
Price[perfume, type] := (7, 18,
                               6, 14); [$ / oz]
```

....

#### VARIABLES

```
Production[perfume, type]; [oz]
RawMat; [lb]
...
```

#### SUBJECT TO

```
Production = RawMatYield * RawMat
```

$$[oz] = \frac{[oz]}{[lb]} [lb]$$

Invalid operations among indices can easily be detected, as each set in the model has an assigned unit. The system checks automatically the unit of functions and constraints, and applies unit conversion if necessary.

Another two modelling languages that have taken into account the measurement of units are AIMMS and LPL. LPL extends the declaration of numerical entities by indicating explicitly the units. This practice increases the reliability and the readability, relying on the compiler to check and find syntax errors associated with unit consistency without the user intervention. Every expression is checked for unit imbalance before it is evaluated, and the unit conversion takes place as required. LPL extends the declaration of the numerical entities by introducing the reserved word UNIT and adding a unit expression. AIMMS provides the ability to associate units of measurement with model identifiers that assure unit consistency within expressions.

A useful aid in achieving an automated form of unit consistency is to include a *units conversion table*.<sup>24</sup> A common error in unit consistency is the misuse of numbers of magnitudes, such as t and Kt, g and Kg. In these cases either the order of magnitude is wrong or the system is wrong.

AIMMS also offers the facility of so-called “unit conventions”, which offer a global medium to specify alternative (scaled) units for multiple quantities, units and identifiers. Once the user has selected a particular unit convention, AIMMS will interpret all data transfer with an external component (read, write, display, solve) according to the units that are specified in the convention. The same application can then be used at different sides of the ocean by merely selecting the appropriate convention.

Despite the usefulness of this facility, the majority of current modelling languages do not provide this feature.

### Time-based Modelling

Time plays an important role in various real-life modelling applications where models are solved repeatedly as time passes. Typical examples are found in the area of planning and scheduling. A large portion of the data in time-dependent models originates from the real world where quantities are specified relative to some calendar. Optimisation models usually refer to abstract model periods, allowing the

optimisation model to be formulated independent of real time. This common distinction makes it essential that quantities associated with real calendar time can be converted to quantities associated with model periods and vice versa.

In the modelling language AIMMS there is support for special data types for time-based modelling, namely “calendar”, “horizon”, and “timetable”. A timetable is an indexed set, which links model periods in a planning horizon to time slots in a calendar. By changing the current data (a time slot), an application can be made to solve repeatedly while moving through calendar time.

### ***Procedural Features***

Procedural statements arise from the necessity to specify certain classes of models that cannot be otherwise achieved using only the declarative features of an MP language. The stages in the optimisation process can benefit from procedural statements. When the language is not purely declarative, these procedural features can be used to control the flow of the program.

#### **Procedural features within the algebraic language**

- *IF – THEN (– ELSE) expression*

This is a control flow mechanism. It provides a logical structure, and pathways specifying the execution of groups of expressions. It also adds clarity to the model, as the modeller can group the constraints that only differ in the way they are executed. It therefore also reduces the number of constraints to be specified explicitly in the model.

subject to InventoryBalance {p in PRODUCT, f in FACTORY, t in TIMEPERIOD} :

$$\text{ProdQty}[p,f,t] + (\text{if } t=1 \text{ then InitialInventory}[p,f] \\ \text{else Inventory}[p,f,\text{prev}(t)]) = \text{Inventory}[p,f,t] + \text{sum } \{d \text{ in DEALER}\} \text{SendQty}[p,f,d,t] ;$$

In this example representing the inventory balance in a supply chain problem the use of *if-then-else* expression makes it possible to write only one constraint for all time periods, rather than several constraints for the different cases of the time period t. The summation on the left-hand side of the equation is done over different elements depending on whether  $t = 1$  or  $t \neq 1$ .

- *LOOP (WHILE and FOR) statements*

WHILE and FOR statements are used to loop over a group of statements taking into account certain condition(s) that have to be true for the loop to be executed.

### Procedural features outside the algebraic language

These statements are used separately from the model definition. These features are important when the language is used to “process” model classes that require close coupling in the modelling and solving steps.

- *SOLVE, INCLUDE, PRINT, ...commands*

These language extensions carry out actions that relate the model with the optimisation environment. It is achieved through a context that permits the use of different commands, from the specification of when to solve the model, to the selection of the type of print out that the analyst requires.

- FOR, WHILE, REPEAT, and IF statements are used as commands, which allow compact iterative processes. The utility of this feature resides in the possibility to carry out instructions and commands related to the same model instance as many times as she (he) needs to execute them. This feature facilitates the use of a generic structure, as it allows repeated optimisation of the same problem with different data sets. Equally important is the facility to solve a sequence of models where a model gets as input the output of another model.

AMPL permits the inclusion of a small program (or script) that is to be executed. Such a script allows the above looping commands, a solve or a call to any other script. AIMMS uses procedures for the same purpose, where these procedures can have arguments as well.

### **Logic**

*Combinatorial Optimisation* problems frequently occur in real life, in cases such as crew scheduling, generalised assignment, and timetabling. In such problems a set of decisions with binary choices or even multiple choices have to be made, and a set of constraints connecting the decisions has to be satisfied.<sup>39</sup>

The extensive research on these problems has proven that Integer Programming / Mixed Integer Programming (IP / MIP) is a suitable approach for formulating and solving a wide range of industrial problems.<sup>40</sup> However, this cannot be assured in all circumstances. Although the transformation needed to go from the problem

conceptualisation into the IP / MIP algebraic model specification can be done by using a reformulation procedure,<sup>41</sup> this may become complex. When applying such a procedure sometimes it is necessary to introduce problem-specific modifications in the formulation as well as in the solving procedure.<sup>42</sup> Therefore, for the MP systems to perform satisfactorily in a broader scope of the combinatorial optimisation problems, the syntax of the MP languages and the model representation features have been extended. Solution algorithms which process these discrete / logical formulations are also similarly extended. Fourer<sup>42</sup> points out the principle of applicability, independence, consistency, generality, and implementability that such modelling extensions should follow. Some of the current MP modelling languages that have considered this extension in their syntax are MPL<sup>41</sup> and AMPL, and more recently OPL.<sup>43</sup>

Some work on extending the algebraic modelling languages to facilitate the formulation of such logic features within an MP framework include the following:

- (i) extensions to the algebraic modelling languages using logical operators, conditional operators, and counting operators,
- (ii) introduction of predicate calculus constructs, such as *and*, *or*, *at\_least*, *at\_most*, *all\_different*, which are combined with algebraic statements,
- (iii) specification of variables used as subscripts. These extensions avoid the use of the 0-1 variables introduced to suit combinatorial constraints. Notably AMPL has pioneered this approach.

An alternative way of representing such combined algebraic and logical restrictions is based on the well-established modelling approach of logic programming. *Constraint (Logic) Programming* (CLP) uses first order logic, particularly propositional and predicate calculus, and seeks to satisfy linear constraints within a logic programming framework. Some CLP typical systems are CHIP,<sup>44</sup> Prolog III,<sup>45</sup> and ECLiPSe.<sup>46</sup> The syntax used in a CLP model representation can be similar to the problem owner's conceptualisation of the problem; a very useful feature provided by these systems.

CLP often leads to a very compact representation of the given problem, however, the solution process, which is essentially a tree enumeration method differs from the IP solution algorithms.

## Optimisation Models under Uncertainty

Although Stochastic Programming (SP) was introduced early on by Dantzig and Mandansky,<sup>47,48</sup> in recent times it has seen considerable interest and growth in application. In SP models two important modelling paradigms (i) Optimisation and (ii) Models of Randomness are brought together.

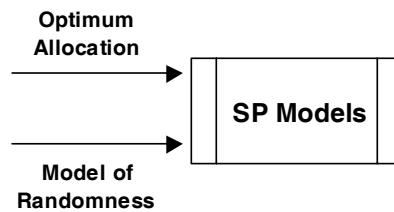


Figure 11 SP Modelling Paradigm

There are data standards, such as SMPS,<sup>5</sup> which include naturally the time, and scenarios to capture the randomness. Modelling languages need to extend their syntax, to support special structures that occur in this combined modelling paradigm.<sup>50</sup>

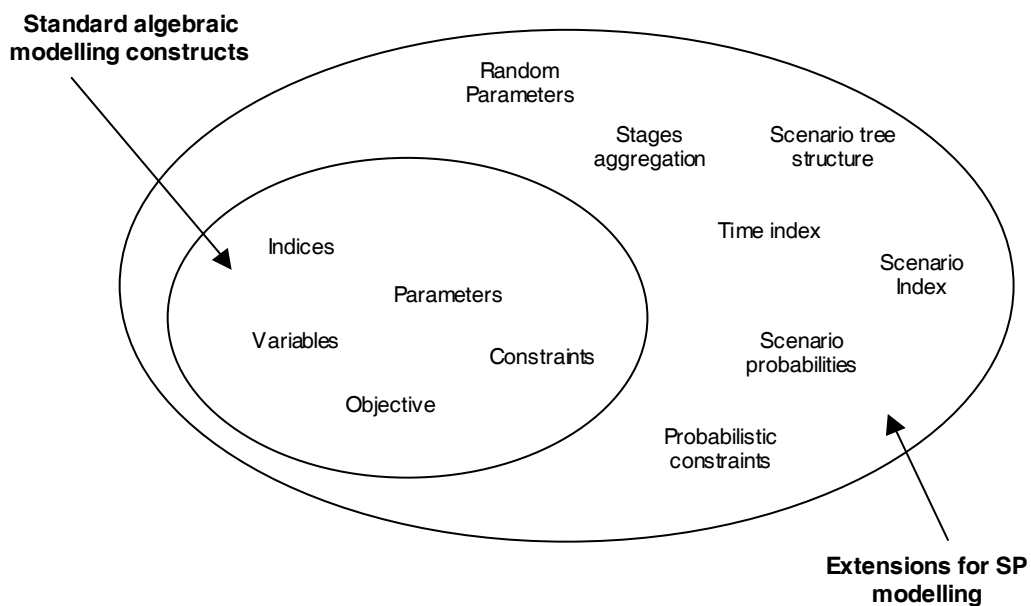


Figure 12 Extended Language Constructs

The items of the stochastic information, which are captured by such syntax extensions are summarised below: (i) *Stochastic class*: identifies the category to which the SP problem belongs, such as distribution-based recourse problem, scenario-based recourse problem, and chance-constrained problem;<sup>49</sup> (ii) *Scenario tree*: for scenario-

based problems, it represents the structure of the event tree; (iii) *Stages*: the time horizon of the underlying dynamic linear program can be partitioned into decisional stages; (iv) *Scenarios probability distribution*: the (discrete) probabilities associated with the scenarios; (v) *Random parameters probability distributions*: only required for distribution-based problems; (vi) *Scenario dimension*: only required for scenario-based problems; (vii) *Time dimension*: the index is used to describe the temporal horizon in the underlying model needs to be uniquely identified; (viii) *Random data*: defines and marks the random parameters of the problem.<sup>72</sup>

A number of alternative modelling and solution systems have been proposed. These are set out in Table 3:

<b>SYSTEM</b>	<b>Researchers</b>	<b>Affiliation</b>
SPOSL	King et al.	IBM
SPInE	Mitra et al.	Brunel University/OptiRisk Systems
SLP-IOR	Kall et al.	University of Zurich
STOCHGEN	Dempster et al.	University of Cambridge
SP-AMPL	Fourer R.	Northwestern University
MSLiP	Gassman et al.	Dalhousie University
SETSTOCH	Frangniere et al.	University of Geneva
DECIS	Infanger et al.	Stanford University

*Table 3 Systems for Stochastic Programming*

For not very large instances of stochastic problems it is possible to create, using algebraic constructs, a deterministic equivalent problem for the given SP problem. For realistic cases, this is nearly impossible. Due to the constraints added to express the dependencies between variables of different stages the number of constraints grows dramatically, and the model is tractable only up to a limited number of variables and constraints. To be able to solve such large cases it is necessary to express the SP model compactly and couple it closely with the solver, which typically applies a decomposition procedure.

## Coupling of Modelling and Solving

MP modelling and solution systems have coevolved over the last two decades. In the late 70s Geoffrion<sup>51</sup> made a strong case for separating modelling and solving. One major argument in favour of this separation was to make the underlying structure and the models more transparent to the different constituents who participate in the development of such decision models. Recently, there has been a strong trend towards achieving a closer coupling between the modelling stage and solution process. This development is not necessarily in conflict with the ‘separation’ principle. The models are developed in a form that is easily communicated between analysts. However, by having a closer coupling with the solver, the solution process is often improved in many situations. This is an important development and in the following sections we present a few modelling and solving contexts in which this situation is applied.

### **Column Generation**

A number of production problems in industry involve designing cutting patterns to turn raw materials into product parts. These arise in industries such as the production of paper, textiles, wooden/metallic boards, and gaskets. While the objective may be to minimise wastage or raw material supply, the key issue is the construction of all the alternative patterns by which the raw material can be cut to produce the different products. In many of these industries the resulting number of cutting patterns is enormous, and consequently the resulting MIP is very large and difficult to solve.

Gilmore & Gomory introduced the *Column Generation* technique in 1961.<sup>9,52</sup> The cutting stock problem can be stated as

$$\begin{aligned} & \text{Min} \sum_j c_j x_j \\ & \text{s.t.} \sum_i a_{ij} x_j \geq b_i \end{aligned}$$

where  $b$  is the demand for different parts, and the columns,  $j$ , of  $A$  represent the number of parts  $i$  produced if cutting pattern  $j$  is used. By considering the pricing stage in the simplex method, we choose as the entering column

$$\text{Min}_j c_j - \sum_i a_{ij} \pi_i$$

Where  $\pi_i$  denotes the shadow price associated with the  $i^{th}$  constraint.

Since  $c_j$  is a constant, this amounts to first solving the following knapsack problem, where  $W$  is the width of the raw material and  $\pi_i$  are the current optimal shadow prices.

$$\text{Max } \sum_i a_i \pi_i$$

$$\text{s.t. } \sum_i a_i \leq W$$

$$a_i \geq 0 \text{ and integer}$$

The cutting stock procedure is outlined below:

1. Select a subgroup of columns (variables) from the total available cutting patterns. A heuristic procedure can be used for this purpose and solve the cutting stock problem.
2. Determine the new cutting pattern by solving the knapsack problem. If a new column is not found stop, else return to step 1.

An alternative but usually less computationally efficient approach is *Column Evaluation*. In this method all possible cutting patterns are generated. This approach is similar to column generation in that once the model is initially solved, in future runs we compute the reduced costs of every cutting pattern and only include those patterns that price out favourably in the resulting cutting stock optimisation problem.

Finally, column generation is also introduced in those modelling situations where it is difficult to capture the structure of the problem in a declarative fashion. This often arises in scheduling problems where different rules determine in which constraints the entries of a column are made. The resulting problems are large MIP problems involving special ordered sets of type 1. This feature is then exploited in the optimisation phase.

### ***Rounding Heuristic and Branch & Price Approaches***

The MIP problems are often difficult to solve. To gain insights into these problems it is often desirable to relax the integer requirement and examine the resulting LP. In this way we can incorporate filters into the modelling stage by which integer variables can be fixed or removed from the problem. This is achieved by analysing the relaxed LP solution to the original optimisation problem. Although this often results in sub-

optimal solutions these can then be introduced as bounds into the original unrestricted optimisation problem.

Progressing from a relaxed LP solution to an integer solution is not easy.<sup>53</sup> When the scope of final widths (number of rows in the matrix coefficients) is narrow, the heuristic approach for rounding (up or down) each fractional valued variable to the nearest integer works efficiently. The difficulty arises in problems with a wide scope of final widths, where the approach of rounding to the nearest integer can result in excessive increases in the objective function value.

A typical heuristic approach for rounding may be summarised in the following steps:

1. We relax the integrality of the variables, and solve the relaxed LP, obtaining a solution  $\bar{x}$ .
2. We take the original IP problem, and modify it in the following way:

We include  $\bar{x}$  (solution of the LP) as data parameters in the IP.

We establish rounding conditions according to the solution obtained from the LP problem:

Variables with  $\bar{x}$  integer (except the variables with value zero) are excluded from the optimisation process, as they are set to this integer value. Variables with a value  $\bar{x}$  close to an integer value are rounded to it, as they are likely to have a very similar result. Therefore, this rounding will not increment dramatically the cost.

3. Repeat the previous step until the stopping criteria has been satisfied.

This heuristic is an iterative process decomposed into solving two models. Ideally the problem owner does not have to change the model every time and he can specify the starting basis for the iterations, the rounding approximations that are considered in the IP after the LP has been solved, and the stopping criteria. Through a script or an interactive interface the solving process can benefit from a modelling environment that lets the user specify procedural steps that are specific to each application.

### **Branch and price**

Column generation is applied to the LP relaxation, whereby the IP contains only variables that priced out positively in the LP relaxation solution.

The typical steps of the branch & price approach are set out below:

1. Generate the LP relaxation with a subset of all the columns.
2. Solve the 'pricing problem' (a separation problem from the dual LP) to identify columns to enter the basis.
3. Solve the LP relaxation with the new columns.
4. Carry out branching when no columns price out to enter the basis, and the LP does not satisfy the integrality conditions.

### ***Scenario-Based SP***

In Stochastic Programming we bring together models of randomness for the uncertain parameters and models for optimum resource allocation. By generating scenarios the random parameters are investigated. There are two procedures in which modelling and solving coupling can be addressed; these are *scenario analysis* and *SP analysis*.

### **Scenario Analysis**

This technique, along with many others, arises from the need to get more accuracy and/or completeness in the information obtained from the deterministic models. In this approach different scenarios, that is certain combinations of possible values of the uncertain parameters, are considered. Hereafter the problem is solved for each scenario considered previously. Thus, by solving the problem repeatedly for different scenarios and studying the solutions obtained, the decision-maker observes the sensitivities and decides on an appropriate solution. The main ideas of scenario analysis are as follows.

- (i) To study the model by solving the model and analysing the problem for each demand realisation. This is called *Scenario Analysis*.

or

- (ii) To select a subset of scenarios to carry out the analysis. Such selection is done by a heuristic approach. This applies when the model dimensions for a single scenario instance are large and consequently the computational processing time excessive.

As in the case for column generation, a major interaction with the solver makes the process simpler and faster. The use of scripts or an interface that lets the problem owner specify the steps in the solution analysis improves the performance of the application. The core model does not change and the sequence of scenarios which are applied as data instances to the model are specified concisely.

### **Analysis EVPI, Scenario Sampling**

To directly address the inadequacy of the deterministic models and to obtain a hedged solution under randomness, SP models are used. In SP we consider the distributions of the uncertain parameters to be known, and use it to generate discrete scenarios which represent the random parameters. Therefore the model size grows enormously, and often becomes intractable. To overcome this difficulty some sampling approaches have been developed. Sampling methods are required to reduce the large dimensionality of realistic problems.

The main questions regarding scenarios are how many and which ones are chosen. To answer these questions different sampling approaches have been presented based on Monte Carlo sampling,<sup>54</sup> and on the expected value of perfect information (EVPI).<sup>55</sup> In these approaches, the iterative optimisation procedure is carried out exploring an increasing number of scenarios until some stopping criteria are satisfied by the solution obtained so far. In these procedures new LP subproblems are repeatedly generated and solved improving progressively the accuracy of the current solution. Modelling, solving, and solution analysis closely interact during this process.<sup>56</sup>

Some work has been done using sequential importance sampling heuristics to approximate the true EVPI and therefore the optimal solution of the problem. The heuristics can be used in a variety of ways to give a whole family of EVPI sampling algorithms.<sup>55</sup>

In all these cases the close coupling between modelling and solving will allow us to develop a good computational and analytic framework

### ***Logic driven solutions in constraints***

Logic programming provides a very powerful and expressive way of representing a decision model. For the last ten years the development of CLP has introduced arithmetical constraints within an otherwise logic programming framework.

Two well-known systems, namely ILOG Solver and ECLiPSe, not only support the modeller in representing the problem, but also to interact and control the enumerative steps.

ILOG Solver is a set of C++ class libraries which can be embedded in users' applications. It can be used independently (as a toolkit) or linked directly to a modelling language (ILOG OPL Studio, AMPL) to act as an interface for a non-expert modeller for solving CLP models. ILOG Solver provides a default search strategy, although it suggests alternatives, and allows the user to create customised search strategies. The user is able to communicate with the solver in an interactive way. ILOG Solver allows the problem owner to enrich the search strategy by modifying or adding constraints that may speed up the solver process due to an improvement in the search space.

Typically, there are explicit search commands, such as (i) ordering by rules, (ii) interleaved depth-first search, (iii) limited discrepancy search, and others including a user specified customised search.<sup>43</sup>

Both ILOG Solver and ECLiPSe allow the user to combine CP search with LP (MIP relaxed to LP and then used to guide the search). Compared to straight processing by MIP guided CLP search can achieve superior computational performance in many classes of models and scheduling problems in particular. The aspects of solving such combined CP and MP models are discussed in <sup>57,58,59</sup>.

### ***Decomposition Algorithms***

Considerable research work has taken place to develop solution algorithms to process large scale linear and integer programming problems with embedded structures. Most well-known of these approaches are Dantzig-Wolfe decomposition,<sup>60</sup> Lagrangean decomposition, and Benders' decomposition.

Dantzig-Wolfe decomposition is mainly applied to LP, and the solution steps require solving independent LP subproblems, and then communicating the primal and dual solution values to an overarching master problem.

Lagrangean decomposition is applied to both continuous LP and IP problems, and also exploits the model structure. In this approach subsets of constraints are aggregated and then weighted by multipliers (Lagrangean), and introduced in the

objective function. Again a series of subproblems is created and solved with converging upper and lower bounds on the objective function.

In Benders' decomposition the model is partitioned depending on the context of the algorithm to an integer part (MIP problems) or first-stage constraints (SP problems). In general the first partition of the problem is used as the master problem. Subproblems from the remaining partitions are then solved, and results are used to update and augment (add cuts) the master problem.

These algorithms are often implemented for specialist applications. By having a close coupling between the modelling system and the solver these classes of algorithms can be investigated.

## Model and Results Analysis

In the course of formulating and investigating optimisation models different analytic steps are applied. These are also known as prototyping and validation procedures, and they are used to refine the model and establish its validity as well as its connection to the physical problem under construction.

The steps of this investigation may be broadly summarised as (a) model analysis, (b) model and data validation, (c) data instantiation and optimisation, and (d) solution analysis and reporting.

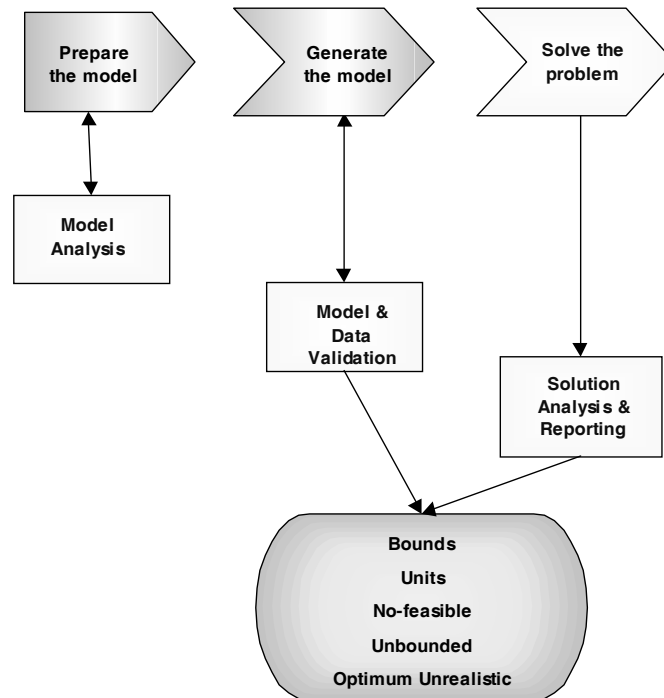
This breakdown of steps is important, and they are not necessarily always applied in one single sequence. The step that involves optimisation can be looked upon as an ‘inferencing’ step of the model for a given data set instantiating the model.

In Table 4 we consider a detailed breakdown of these steps.

Analytical Steps	Analysis Procedures
Model analysis	Syntax checking
	Unit consistency
	Pre-solution analysis (bounds analysis, constraints analysis)
Model & data validation	Model validation (exercise the model for extreme values of data sets)
	Post-optimal analysis (shadow prices analysis, unbounded, infeasible, optimum unrealistic diagnostics)
Data instantiation & optimisation	Solve the model with the “proper” data instance, that is, realistic or actual data sets
Solution analysis & reporting	What-if analysis
	Scenario analysis
	Diagnostics based on logical-consistency

*Table 4 Prototyping and Validation Procedures*

In Figure 13 we illustrate the three main analyses, that are model analysis, model and data validation, and solution analysis, and the stages when optimisation is executed.



*Figure 13 Analyses in the Modelling Process*

### **Model Analysis**

*Syntax error checking* can detect ‘early’ anomalies in the model formulation. The generation of a machine-readable form of the model cannot take place until the specification is syntactically correct. By using a syntax checker, errors (e.g. typographical mistakes) are detected before the model is completely formulated. Domain and range errors (e.g. the modeller might have forgotten an index of a variable) are also detected before the completion of the model formulation. If it is available as a language feature *unit consistency* checking should be applied; this step is extremely valuable as it often identifies (a) conceptual difficulties and (b) inconsistencies in respect of units of measure used in different groups of data within a given model data set.

Once the model is syntactically correct, logical analyses of the model can be carried out to detect unbounded, infeasible or a potentially unrealistic optimal solution.<sup>39</sup> The techniques used for analysing the model at this stage are usually called pre-solution analysis. McCarl<sup>61</sup> presents a series of *pre-solution analysis* rules, some of which have been previously presented as computerised techniques,<sup>62</sup> Greenberg<sup>63</sup> in ANALYZE, and Chinneck<sup>64,65</sup> in MProbe, among others. These pre-solution analyses

in general aim to find “problematic structures” by examining the matrix structure, and by establishing bounds on constraints. DOME<sup>24</sup> provides the option for pre-solution analysis through a script driver.

### ***Model and Data Validation***

Before carrying out optimisation an analyst would normally check the logical consistency within the model. In supply chain models production flow balances can be typically checked (e.g., everything that is produced, is either sold or stored). With the *model validation* the analyst checks whether or not the model makes sense (e.g., if production sites are closed then there is no production). There is a need to create simple procedures to establish that the model is robust, that is, it provides logical and acceptable results when it is exercised using data sets with extreme values. The model validation indicates the degree to which its results should be believed, and is used to gain confidence in the model.

In practice once the optimisation study has been carried out there are two major components which surge as pertinent to the analyst. These are the model and the results. The analyst looks at these two elements from two different perspectives: Diagnostics and Reporting.

For diagnostic purposes, the model and the solution can be seen as an entity, or as two separated items. By studying and analysing both, the analyst can investigate any of the three possible diagnostics, which are ‘unbounded’, ‘infeasible’, and ‘optimum unrealistic’. These outcomes are observed as a consequence of some mismatch in the model and data combination.

Optimisation models and data instances are difficult to separate when dealing with the results, and have a combined impact on the results (‘outcomes’). Therefore, model and data debugging are carried out to diagnose the “illness” of the model formulation and solution. *Post-solution techniques* are used to capture problematic structures that lead to an infeasible, an unbounded, or an unrealistic solution. These techniques are usually based on analysing shadow prices, sensitivity analysis and/or reconstruction of constraint calculations. Many of these post-solution techniques can be applied manually, however, GAMSCHK<sup>61</sup> provides an excellent automated approach.

### ***Solution Analysis and Reporting***

After model diagnosis, the analyst may carry out “*what-if*” *analyses*, where the decision-maker changes the input values, using different model data instances. Sharda & Steiger<sup>66</sup> call this technique “model analysis”. This technique examines the changes of the optimal solution and the optimal objective value with respect to variations in some of the parameters that are considered to be important. It is usually done by varying one parameter at a time. Another technique that varies uncertain parameters is *scenario analysis*. In this approach different scenarios, that is certain combinations of possible values of the uncertain parameters, are considered. Hereafter the problem is then solved for each of these scenarios. Thus, by solving the problem repeatedly for different scenarios and studying the solutions obtained, the decision-maker observes the sensitivities and decides on an appropriate solution by following a heuristic approach.

A further analysis that may be carried out is the *exogenous logical-consistency* for external requirements that cannot be captured by the optimisation model. Typical examples are complex taxation rules, regulatory requirements and others, which were not considered in the model formulation. A domain experts knowledge is required to decide whether or not the optimal solution is realistic.<sup>11</sup>

For reporting purposes, the model and the solution are seen as items to be presented to the decision-maker in a meaningful way. Reports are as important as the previous analyses carried out. Clear and concise reports facilitate the extraction of relevant knowledge and aid decision making. Such report preparation and representation is well supported by OLAP tools,<sup>67</sup> which include data manipulation functions.

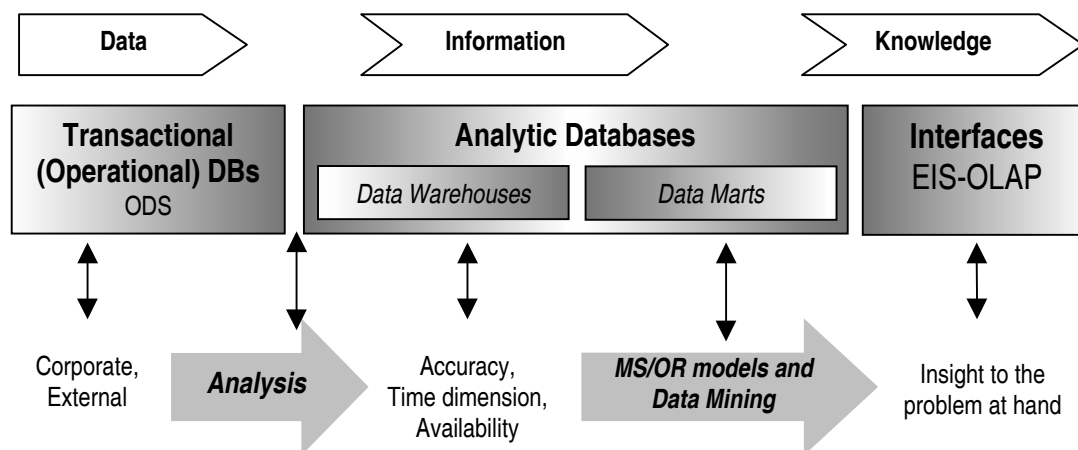
The modelling language AIMMS offers facilities to build an interactive report for the end-users of an application. Such a report consists of pages with tabular and graphical objects that allow for two-way data communication. Navigation through these pages goes through buttons, menus and dedicated navigation objects. There is also a facility to specify graphical print pages similar to regular pages, but with the extra functionality that they can be printed as a complete report.

## Positioning within the IS Architecture

### *The Information Value Chain (IVC)*

Model-based decision-making is well accepted within the organisational context of the industrial, commercial, and the public sector organisations. Mathematical Programming Modelling supports optimum decision-making in these domains.

Quantitative (management) models constitute an important component within the information value chain (IVC) (see Figure 14, and <sup>22</sup>). The analysis and the synthesis of transactional (operational) data create analytic data or information, which is the first step in the information value chain. In general the collection of analytic data items or information relating to an organisation is called a Data Warehouse. The information residing within a data Warehouse is used by decision-makers and knowledge workers across the organisation.



*Figure 14 Information Value Chain*

An organisation has typically many divisions, such as accounting, marketing, sales, production, and distribution. In a composite decision making environment the organisational Data Warehouse is used across all divisions. A decision problem may relate to an individual department, in which case we extract the analytic data from the Data Warehouse pertinent to that specific department. This subset of information is known as a Data Mart.

Knowledge workers require a variety of ways in which they can interact with, and exploit the available information. They may need to aggregate data and make graphical displays of summarised data of the information residing within a data Mart, or to look for patterns. A range of modern database tools known as Online Analytical

Processing (OLAP) have been created which enable decision makers to work directly with Data Warehouses and Data Marts.<sup>67,22</sup> Because of their multidimensional characteristics, they also help internally to manipulate the data, and to visualise the data items residing in Data Marts and Data Warehouses.

Typically, information taken from Data Marts is used to instantiate a range of models, which are developed to provide valuable insight to the problem owner. Such models either ‘describe’ a problem at hand, or ‘prescribe’ a set of possible, yet alternative solutions. Simulation for instance is a typical example of a *descriptive model*, while optimisation-based models are examples of *prescriptive models*, that is, in an ideal world of abstract representations these models can only “prescribe” decisions for particular problems.

In a given problem domain the study of models and solution results, leads to ‘knowledge’ for the decision-maker. Thus with repeated application of descriptive and prescriptive models, the decision-maker gains valuable insight, that is knowledge about the application. Further, this knowledge is ‘dynamic’ in that it continuously evolves as the information for the problem changes, and the problem owner re-applies the models to understand its nature and behaviour. An overview of the information value chain is set out in Figure 14.

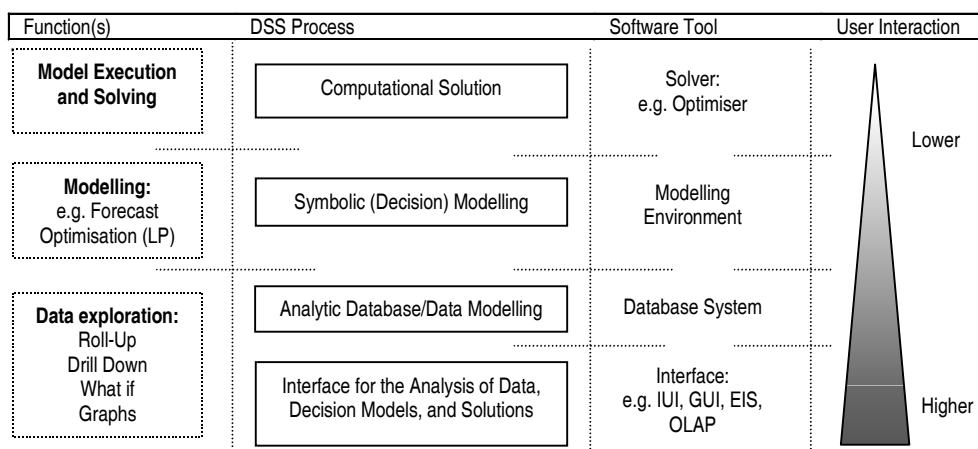
### ***Data and Symbolic Modelling***

*Data (-base) Modelling*, *Decision Modelling*, and *Model Investigation*, are logical constructs, which play a leading role both in the interaction of information systems and decision technologies, as well as in rational decision making. It is easily seen that Data Modelling and Decision modelling closely interact with each other. The following describes these logical constructs and their relationship to knowledge.

1. **Data Modelling** refers to the ‘structured’ internal representation and external presentation of recorded facts. Broadly speaking this provides the decision-maker with information about their decision problem.
2. **Decision Modelling** is the development of a model, or a range of models that captures the structure as well as the decisions in respect of a given problem. These models are used to evaluate possible decisions (actions) in a given problem domain, and the probable outcomes of these actions.

3. **Model Analysis and Investigation** refers to the instantiation of the model with data, and the evaluation of the model parameters as well as the results in order to gain confidence and insight into the model.

Typically, data modelling involves defining relationships between data items leading to a relational data model, or identifying categories that are then used to define multidimensional tables, leading to a multidimensional data model. Symbolic or Decision modelling involves the development of models that are used for decision-making. Classical decision models include linear, non-linear, or discrete optimisation models. However, we consider simulation, cluster analysis, forecasting, data mining, and other analytic models equally appropriate for decision support. Model analysis and investigation is often a descriptive analysis of the results obtained which is applied to gain insight, or knowledge with respect to a given decision problem. In particular, during the analysis phase, a series of ‘what-if’ investigations are used to change parameters and, therefore, to evaluate the understanding of the problem by the decision-maker. We consider the required *functions*, *DSS processes*, *software tools*, and the level of *user interaction* in Figure 15.

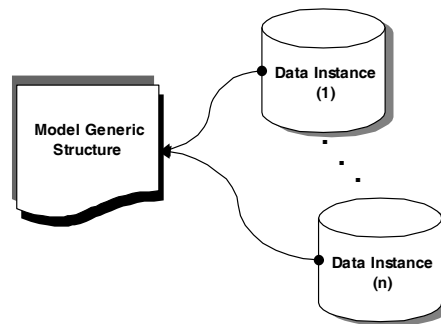


*Figure 15 Relative positioning of Tools, DSS functions, and User Interaction.*

Figure 15 sets out the hierarchy of tools involved in the overall decision process and their relative positioning at each system level. A complex decision process has many constituents. At the lower level the decision-maker uses the system to ‘guide’ him in his decisions. At the middle level, the database and model specialists are responsible, respectively, for maintaining and developing the data instances and the decision model. At the higher level, solving experts are responsible for obtaining computational solutions of the instantiated models. Naturally, the problem owner

interacts most with the system at the outer level. The degree of end-user interaction is gradually reduced at the lower levels, and is the least at the innermost level; that is obtaining computational solutions to the models implemented. This framework is analogous to the concept of different views put forward by Greenberg.<sup>63</sup>

Geoffrion<sup>33</sup> points out the importance of having a generic structure, which leads us to store separately different data instances (Figure 16).



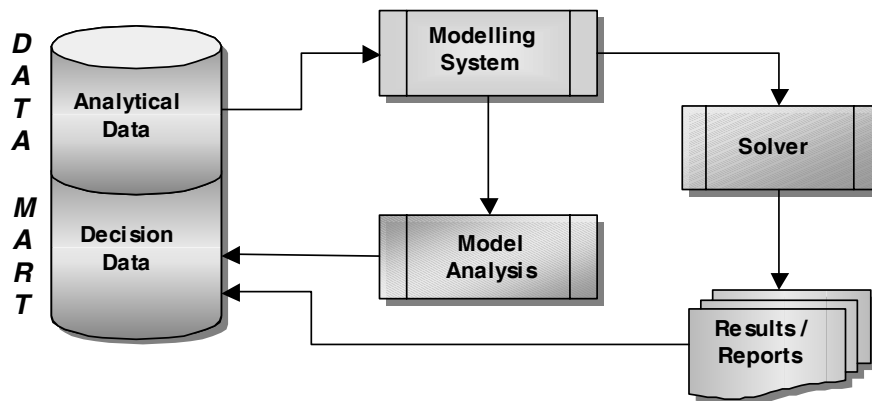
*Figure 16 Generic structure and Data Instances*

A generic representation of the decision problem is desirable and this is achieved through the interaction between data modelling and symbolic modelling. In most MP models there is a direct mapping between the sets and indices that form the decision model and the structure of the data used to instantiate the decision model. This close coupling and physical separation of data modelling and decision modelling leads to a number of distinct advantages for the implementation and use of decision support systems:

- (i) Large amounts of data are managed and maintained by database systems, which are built for this purpose, instead of MP modelling systems.
- (ii) Decision models can be variously instantiated with the use of different data marts.
- (iii) The information used to develop the decision support system is normally available in corporate databases. Thus it is easier to extract the necessary information from corporate systems and use it directly in the corresponding DSS, avoiding the development of proprietary data collections and connections to the DSS.
- (iv) Database Systems, and Data Marts in particular, are easily connected to a range of analytical tools which can facilitate the analysis of optimisation results.

Therefore analysis of optimisation results is easier to achieve by using dedicated data marts, and standard database technology.

In modern MP modelling systems, the connection between modelling languages and the database/spreadsheet is in both directions: retrieving data from a database to instantiate the decision models, as well as exporting data into the database after solving the models (Figure 17.)



*Figure 17 Model & Data Interaction*

However, the connectivity to database systems has only recently become a standard feature in MP Modelling systems. One of the first modelling languages to incorporate database connectivity was MPL.<sup>68</sup> However, most current algebraic modelling languages provide this facility (see AIMMS, AMPL, GAMS, LINGO, MPL) and can also access data that is stored in data files, spreadsheets and/or most commercial databases. In addition, some modelling languages, such as AIMMS, AMPL Plus, LINGO, and MPL, retrieve and store data through Open Data Base Connectivity (ODBC).

### ***The Decision Database***

It is easily seen that for any given decision support system, data models, symbolic models, and algorithmic tools interact. This inherently strong coupling of data models and algorithms leads to an inference engine, known as the *decision database*.<sup>21</sup> Alternatively, one can consider the decision database as a *Data Mart*, in which the data models provide information (facts) about the problem at hand, decision models describe the decision problem, or the known relationships between the known information, and the algorithms generate the set of optimal, or near optimal “solutions” for the problem at hand.

We have introduced the Data Mart earlier; within the context of a decision database, a model-oriented view of a Data Mart can be set out as a compact, application-specific analytical database. In this data mart the available information about the decision problem is classified into different groups according to their content.<sup>22</sup> In stochastic programming applications, for instance, the decision database may consist of four main types of information. *Deterministic* Information (or information invariable to the uncertainty of the future), *Stochastic* Information (possible characteristics of future uncertainty and the information that is subject to that uncertainty), as well as the *Solutions* (possible courses of action in the uncertain future). Further, it will include different solutions from different solving methods for a particular problem (output results, e.g., ‘wait-and-see’ scenario analysis, and ‘here-and-now’ solutions).

The coexistence of analytical data, decision model, and decision data comes naturally, as they share the same structure. This structure also imposes the natural coupling of data modelling, symbolic modelling, and the model analysis. This leads us to consider modern optimisation-based DSS as application-specific information systems, consisting of Database Tools, Algebraic Modelling Tools, and Algorithmic (‘solving’) tools, with additional analytical tools which are often used to provide further insight into the problem at hand. These additional analytical tools may consist, for example, of (Relational) Online Analytical Processing (R)OLAP tools which can help an analyst achieve a fast and descriptive overview of the data, in order to study the problem, and make appropriate reports.

## **New Directions, Discussions and Conclusions**

The re-emergence of optimisation methods and the growing acceptance of optimisation models for decision support within the business community<sup>69</sup> have fuelled the development and maturing of modelling and solving tools.

Our investigations reveal new developments encompassing data connectivity, language extensions for new modelling paradigms, graphical user interface, application generation and prototyping, and development of web-based decision support systems.

### **Data connectivity**

From the user's point of view spreadsheets, such as Excel, seem to be a comfortable environment for creating optimisation applications. Spreadsheets are broadly utilised by many users as an analytical tool. Furthermore, they incorporate add-ins for optimisation that allow users to develop MP applications within a familiar environment. Consequently spreadsheets are gaining in popularity and importance as optimisation modelling support environment. However, spreadsheets are not always the best option for optimisation modelling, due to the lack of flexibility when dealing with model representation, while the optimisation algorithms available in spreadsheets are often limited in capabilities and features, and do not scale very well. Gass et al.<sup>70</sup> support that spreadsheets should not be considered as adequate environments for optimisation, in comparison to "expert choice" software, such as MP modelling languages and solution algorithm tools, which are targeted towards operations research practitioners.

Nevertheless, the use of databases and spreadsheets for data storage, and the connection of modelling tools to these data sources are well established, and have proven benefits such as model / data independence.<sup>71</sup> Embedding a modelling and solution system within an IS environment and connecting these to database tools is gaining new research interest. By using a database/spreadsheet to store the data the analyst can benefit from data manipulation tools, multidimensional viewing and graphical presentation, such as OLAP.

## **Language extensions for new modelling paradigms**

In recent times two new developments in MP have found a growing number of applications. These are *Constraint Satisfaction (CS)* and *Stochastic Programming (SP)*.

CS constructs are being introduced within algebraic modelling languages such as OPL, AMPL, and MPL.

Similarly, real life problems with some uncertain parameters can be modelled as *Stochastic Programs*, which bring together models of optimum resource allocation and models of randomness. This combination requires special constructs which can represent the interdependencies between time periods, stages and scenarios. SMPL and SAMPL<sup>72</sup> are two examples of extensions to existing algebraic modelling languages which support SP modelling.

## **Graphical User Interface (GUI)**

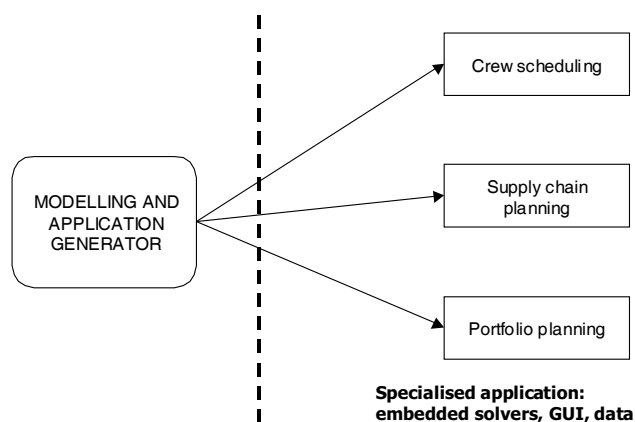
MP Modelling tools have advanced substantially and follow the paradigm of “user-centred design”.<sup>73</sup> Although some modelling languages are still command driven, most MP modelling systems support visual interactive modelling and use Windows-based interfaces. Thus, users can interact easily and flexibly with the modelling tool, and make use of advanced graphical user interface features such as tree structures representing the model definition, and context-sensitive on-line help.

According to Steiger and Sharda,<sup>74</sup> the quality of the user interface is an efficiency measure of an MP modelling system along with compactness of representation, error analysis and correction, and learning difficulty.

## **Application generation and prototyping**

A number of optimisation techniques, such as linear and integer programming, stochastic programming, and constraint logic programming have been widely extended and are accepted as a tool to help in decision making in different areas of industry and commerce. Previously, the main focus of MP software development was computational solution tools (50s and 60s), and the development of modelling languages (in the 70s and 80s). Subsequently the research and development issues converge in the creation of DSS, which are integrated environments for application construction.<sup>23</sup> AIMMS is an example of such an integrated environment. New

environments form an important and competitive research area. However, DSS applications in diverse sectors require constituents with appropriate domain knowledge that needs to be captured in the corresponding models. It is therefore necessary to provide problem owners from different sectors specialised models that best represent their application domains and their decision making requirements. These are also known as vertical applications or domain specific ‘solutions’. Such specialised applications comprise of user interface, database, optimisation models, and embedded solver tools customised for specific application domains, such as crew scheduling, supply chain planning or portfolio planning.



*Figure 18 Application Generation (Optimisation)*

At par with developments in the computing and software industry, component-based systems are emerging as the architecture of choice for MP-based decision support. Two leading examples of component-based DSS applications development tools are OPL Studio,<sup>75</sup> and OptiMax.<sup>26</sup>

### **Web-based DSS**

The Internet has become the chosen platform for conducting business, and a wide range of applications has been developed for processing of business-to-business and business-to-customer commercial transactions. These are commonly called e-commerce solutions.

Taking into consideration (a) the wide acceptance of model-based decision-making within an organisational context and (b) the modularisation and component-based architecture adopted by the optimisation systems developers, it is a natural step forward for the optimisation community to enter the Internet arena.<sup>76</sup> All these

technologies enable the development of MP models,<sup>77</sup> and delivery of decision support solutions via the web. For instance, the NEOS server<sup>78</sup> allows users to submit programs through Web forms, e-mails in a specific format or by using a Java or Unix Tcl/tk specialised interface. The NEOS server is currently connected to more than 25 solvers, and it can be extended, by registering new solvers that may reside anywhere on the Internet. MP-based DSS that provide the services through the Web, so-called Application Service Providers (ASP), keep most of the software on a server, and rent out such services to the users via the Internet. For example, SAS<sup>76</sup> follows a multi-tiered architecture in which the user builds and optimises a model via a standard Web interface, and the processing is distributed across multiple servers. The optimisation server then e-mails the user the results. Another example of an optimisation-based ASP is OSP<sup>79</sup>. OSP intends to provide organisations with the choice to either use optimisation tools and develop and maintain their own models, or to use generic solutions and customise them to match the organisation's applications. In this way it will transform the way organisations develop Optimisation-based Decision Support Systems, in that they will need less specialist skills in-house.

## **Conclusions**

In this paper we have critically reviewed the evolution of tools for Mathematical Programming modelling and their deployment in rapid application development. The early developments of solution technologies led to a need for modelling tools. As these modelling tools developed they in turn inspired further research in the integration of information systems with Mathematical Programming Modelling Systems. Our current work (OSP)<sup>79</sup> has highlighted how support for Mathematical Programming can be provided on the web. We see this as the next exciting research area for delivering Mathematical Programming applications to the end user community.

## **Acknowledgements**

We would like to thank Prof. Johannes Bisschop for his valuable comments and feedback. The authors are grateful to a number of sources for their financial support, which includes UNILEVER Research. We also thank the European Union, which supported Belen Dominguez-Ballesteros (SCHUMANN Project No. 26267, Framework 4); OSP-CRAFT (Framework 5) provides partial support to Dr. Nikitas-Spiros Koutsoukis, and Prof. Gautam Mitra.

## References

- 1 Lucas C and Mitra G (1988). Computer-assisted mathematical programming (modelling) system (CAMPS). *The Computer J* **31**:364-376.
- 2 Fourer R (1983). Modelling Languages versus Matrix Generators for Linear Programming. *ACM Transactions on Mathematical Software* **9**: 143-183.
- 3 IBM World Trade Corporation (1976). IBM Mathematical Programming System Extended/370 (MPSX/370) Program Reference Manual 2<sup>nd</sup> Ed. IBM Publication No. SH19-1095-1: New York and Paris.
- 4 Kuip CAC (1993). Algebraic Languages for Mathematical Programming. *Eur J Opl Res* **67**: 25-51.
- 5 Birge JR, Dempster MAH, Gassmann HI, Gunn EA, King AJ and Wallace SW (1987). A standard input format for multiperiod stochastic linear programs. *Mathematical Programming Society, COAL Newsletter* **17**: 1-19.
- 6 Aho AH, Sethi R and Ullman JD (1986). *Compilers: Principles, Techniques, and Tools*. Addison-Wesley Publishing Company: Reading, Massachusetts, United States.
- 7 Maturana SV (1994). Issues in the design of modelling languages for Mathematical Programming. *Eur J Opl Res* **72**: 243-261.
- 8 Moody S (1994). Methods and tools for modelling linear and integer programming problems. PhD thesis, Department of Mathematics and Statistics, Brunel University.
- 9 Gilmore PC and Gomory RE (1961). A Linear Programming Approach to the Cutting Stock Problem Part I. *Opns Res* **9**: 849-859.
- 10 Mitra G and Darby-Dowman K (1985). Cru-Sched - A Computer Based Bus Crew Scheduling System using Integer Programming. In: Rousseau JM (ed). Proceedings on the 3rd international workshop on transit vehicle and crew scheduling. Computer Scheduling of Public Transport 2, Elsevier Science Publishers BV, Montreal, pp 223-232

- 11 Darby-Dowman K, Fink R, Mitra G and Smith JW (1995). An Intelligent System for US Coast Guard Cutter Scheduling. *Eur J Opl Res* **87**: 574-585.
- 12 Dantzig GB (1951). Maximisation of a Linear Function of Variables Subject to Linear Inequalities. In: TC Koopmans (ed). *Activity Analysis of Production and Allocation*. Wiley: New York.
- 13 Karmarkar NK (1984). A New Polynomial Time Algorithm for Linear Programming. Technical Report, AT&T Bell Laboratories, Murray Hill: New Jersey.
- 14 Mitra G, Andersen J and Levkovitz R (1992). The Interior Point Method for LP on Parallel Computers. In: Kall P (ed). *Systems Modelling and Optimization*. LNCIS180. Springer Verlag, pp 241-250.
- 15 Levkovitz R and Mitra G (1993). Solution of Large-scale Linear Programs: A Review of Hardware, Software and Algorithmic Issues. In: Ciriani T and Leachman RC (eds). *Optimization in Industry, Mathematical Programming and Modelling Techniques in Practice*, John Wiley & Sons Ltd, West Sussex, England, pp 139-171.
- 16 Maros I and Mitra G (1996). Simplex Algorithms. In: Beasley JE (ed). *Advances in Linear and Integer Programming*, Oxford University Press: pp 1-46.
- 17 ILOG CPLEX Division (2000). CPLEX 7.0. URL: <http://www.ilog.com/products/cplex>.
- 18 Ellison EFD, Hajian M, Jones H, Levkovitz R, Maros I, Mitra G and Sayers D (1999). *FortMP Manual, Release 3*. Numerical Algorithms Group (NAG) and Brunel University. URL: <http://www.optirisk-systems.com>.
- 19 Dash Associates (1999). *XPRESS-MP Reference Manual, Release 11*. Dash Associates, U.K.
- 20 Shapiro JF (1998). Bottom-up vs. Top-down Approaches to Supply Chain Management and Modelling. Working Paper #4017, Sloan School of Management, Institute of Technology: Massachusetts.
- 21 Shapiro JF (1993). The Decision Database. Working Paper #3570-93-MSA, Sloan School of Management, Institute of Technology: Massachusetts.

- 22 Koutsoukis NS, Mitra G and Lucas C (1999). Adapting On-Line Analytical Processing (OLAP) for Decision Modelling: The Interaction of Information and Decision Technologies. *Decis Support Sys* **26**: 1-30.
- 23 Mitra G (1989). Tools for Modelling Support and Construction of Optimisation Applications. In: Sharda R, Golden BL, Wasil E, Balci O, Stewart W (eds). *Impacts of Recent Computer Advances on Opns Res*. Elsevier Science Publishing CO. Inc: North Holland, pp 484-496.
- 24 Mousavi H (1998). Data and Optimisation Modelling for Decision Support. PhD thesis, Department of Mathematics and Statistics, Brunel University.
- 25 Koutsoukis NS, Dominguez-Ballesteros B, Lucas C and Mitra G (2000). A Prototype Decision Support System for Strategic Planning under Uncertainty. In: Irani Z, Love PED and Li H. (guest eds). *Special Issue in Supporting Supply Chain Management through an IT/IS Infrastructure (Part II)*. *International Journal of Physical Distribution & Logistic Management* **30**: 640-660.
- 26 Maximal Software Incorporation (1996). *MPL Modelling System Release 4.0*. USA.
- 27 Lindo Systems Inc (1998). *LINGO Modelling System Version 4.0*. USA.
- 28 Ellison EFD and Mitra G (1982). UIMP: User Interface for Mathematical Programming. *ACM Transactions on Mathematical Software* **8**: 229-255.
- 29 Fourer R, Gay DM and Kernighan BW (1993). *AMPL A Modelling Language for Mathematical Programming*. Duxbury Press: Belmont, CA.
- 30 Bisschop JJ and Roelofs M (1999). *AIMMS The Language Reference*. Paragon Decision Technology BV: Haarlem, The Netherlands.
- 31 Brooke A, Kendrick D and Meeraus A (1993). *GAMS: A User's Guide, version 2.25*. Boyd and Fraser Publishers: Danvers, MA.
- 32 Hürlimann T (2002). Modelling Languages: A New Paradigm of Programming. Volume on Modelling Languages and Applications. *Annals of Opns Res* (in press).
- 33 Geoffrion AM (1992). Indexing in Modelling Languages for Mathematical Programming. *Management Science* **38**: 325-344.

- 34 Hürlimann T (1993). LPL: A Modelling Language. Modelling Tools for Decision Support. *University of Fribourg Series in Computer Science* **2**, University of Fribourg, Switzerland.
- 35 Fourer R and Gay DM (1995). Expressing Special Structures in an Algebraic Modelling Language for Mathematical Programming. *ORSA J Comp* **7**: 166-190.
- 36 Zenios S (1990). Integrating Network Optimisation Capabilities into a High Level Modelling Language. *ACM Transactions on Mathematical Software* **16**: 113-142.
- 37 Bradley GH and Clemence DR (1987). A type Calculus for Executable Modelling Languages. *IMA J Mathematics in Management* **1**: 227-291.
- 38 Bisschop JJ (1988). A Functional Description of an Integrated Modelling Software for Mathematical Programming, presented to the *13<sup>th</sup> international Mathematical Programming Symposium*, Tokyo.
- 39 Williams HP (1999). *Model Building in Mathematical Programming, fourth edition*. Wiley: Chichester.
- 40 Kallrath J and Wilson JM (1997). *Business optimisation in mathematical programming*. MacMillan Press Ltd., Houndmills, Basingstoke.
- 41 Mitra G, Lucas C, Moody S and Hadjiconstantinou E (1994). Tools for Reformulating Logical Forms into Zero-One Mixed Integer Programs. *Eur J Opl Res* **72**: 262-276.
- 42 Fourer R (1998). Extending a General-Purpose Algebraic Modelling Language to Combinatorial Optimisation: A Logic Programming Approach. In Woodruff DL (ed). *Advances in Computational and Stochastic Optimization, Logic Programming, and Heuristic Search: Interfaces in Computer Science and Operations Research*. Kluwer Academic Publishers: Dordrecht, The Netherlands, pp 31-74.
- 43 Van Hentenryck P (1999). *The OPL Optimisation Programming Language*. The MIT Press: Cambridge, Mass.
- 44 Dincbas M et al; (1988). The Constraint Logic Programming Language CHIP. *Proceedings of the International Conference on Fifth Generation Computer Systems (FGCS'88)*: Tokyo, Japan, pp 693-702.

- 45 Colmerauer A (1990). An Introduction to Prolog III. *Communications of the ACM*. **33**: 69-90.
- 46 Wallace M, Novello S and Schimpf J (1997). ECLiPSe: A Platform for Constraint Logic Programming. *ICL Sys J* **12(1)**, Issue1.
- 47 Dantzig GB (1955). Linear Programming under Uncertainty. *Management Science* **1**: 197-206.
- 48 Dantzig GB and Madansky A (1961). On the Solution of Two-stage Linear Programs under Uncertainty. *Proceedings of The Fourth Berkeley Symposium on Mathematical Statistics and Probability* **1**: 165-176.
- 49 Charnes A and Cooper WW (1959). Chance-Constrained Programming. *Management Science* **6**: 73-79.
- 50 Gassmann HI and Ireland AM (1996). On the Formulation of Stochastic Linear Programs Using Algebraic Modelling Languages. *Annals of Opns Res* **64**: 83-112.
- 51 Geoffrion AM (1987). An Introduction to Structured Modelling. *Management Science* **33**:547-588.
- 52 Gilmore PC and Gomory RE (1963). A Linear Programming Approach to the Cutting Stock Problem Part II. *Opns Res* **11**: 863-888.
- 53 Woolsey RED (1972). A candle to St. Jude, or Four real world applications of integer programming. *Interfaces* **2**: 20-27.
- 54 Infanger G (1993). *Planning Under Uncertainty. Solving Large-Scale Stochastic Linear Programs*. Boyd and Fraser: Massachusetts.
- 55 Dempster MAH and Thompson RT (1999). EVPI-based importance sampling solution procedures for multistage stochastic linear programs on parallel MIMD architectures. *Annals of Opns Res* **90**: 161-84.
- 56 Messina E and Mitra G (1997). Modelling and analysis of multistage stochastic programming problems: a software environment. *Eur J Opl Res* **101**: 343-359.
- 57 Smith BM, Brailsford SC, Hubbard PM and Williams HP (1997). The Progressive Party Problem: Integer Linear Programming and Constraint Programming Compared. *Constraints, An International Journal* **1:1/2**: 119-138.

- 58 Darby-Dowman K, Little J, Mitra G and Zaffalon M (1997). Constraint Logic Programming and Integer Programming Approaches and Their Collaboration in Solving an Assignment Scheduling Problem. *Constraints, An International Journal* **1**: 245-264.
- 59 Darby-Dowman K and Little J (1998). Properties of Some Combinatorial Optimisation Problems and Their Effect on the Performance of Integer Programming and Constraint Logic Programming. *INFORMS J on Computing* **10**: 276-286.
- 60 Dantzig GB and Wolfe P (1960). Decomposition Principle for Linear Programs. *Opns Res* **8**: 101-111.
- 61 McCarl BA (1998). Repairing Misbehaving Mathematical Programming Models: A Poor Man's Guide, *INFORMS Meeting in Monterey*: Monterey, CA.
- 62 Brearley AL, Mitra G and Williams HP (1975). Analysis of Mathematical Programming Problems Prior to Applying the Simplex Algorithm. *Math Programming* **8**: 54-83.
- 63 Greenberg HJ (1993). Enhancements of ANALYZE: A Computer-Assisted Analysis System for Linear Programming. *ACM Transactions on Mathematical Software* **19**: 233-257.
- 64 Chinneck JW (1997). Feasibility and Viability. In: Gal T and Greenberg HJ (eds). *Recent Advances in Sensitivity Analysis and Parametric Programming*. Kluwer Academic Publishers.
- 65 Chinneck JW (2001). Analyzing Mathematical Programs using Mprobe. Special issue on Modeling Languages. *Annals of Opns Res* **104**: 33-48.
- 66 Sharda R and Steiger DM (1996). Inductive Model Analysis Systems: Enhancing Model Analysis in Decision Support Systems. *Inform Sys Res* **7**: 328-341.
- 67 Codd EF, Codd SB and Salley CT (1993). Providing OLAP (On-Line Analytical Processing) to User-Analyst: an IT Mandate. Codd & Associates. URL: <http://www.salleurl.edu/~augc/codd2.pdf>
- 68 Mitra G, Lucas C, Moody S and Kristjanson B (1995). Sets and Indices in Linear Programming Modelling and their Integration with Relational Data Models. *Computational Optimization and Applications* **4**: 263-283.

- 69 Merrit J (2000). The Best B-Schools. Special MBA Issue (Oct. 2). *Business Week*.
- 70 Gass SH, Hirshfeld DS and Wasil EA (2000). Model World: The Spreadsheets of OR/MS. *Interfaces* **30**: 72-81.
- 71 Atamtürk A, Johnson EL, Linderth JT and Savelsbergh MWP (2000). A relational modelling system for linear and integer programming. *Opns Res* **48**: 846-857.
- 72 Valente P, Mitra G, Poojari CA and Kyriakis T (2002). Software Tools for Stochastic Programming: A Stochastic Programming Integrated Environment SPInE. Technical Report TR/10/01, Brunel University: U.K., also to appear in: Wallace SW and Ziemba WT (eds). *The Applications of Stochastic Programming*, MPS-SIAM Series in Optimisation
- 73 Norman DA and Draper SW (Eds) (1986). *User Centered System Design: New Perspectives in Human Computer Interaction*. Lawrence Erlbaum Associates: Hillsdale, NJ.
- 74 Steiger D and Sharda R (1993). LP Modelling Languages for Personal Computers: A Comparison, Applied Mathematical Programming and Modelling. Mitra G and Maros I (eds). *Annals of Opns Res* **43**: 195-216.
- 75 ILOG Inc. (1999). ILOG OPL Studio User's Manual. URL: <http://www.ilog.com>
- 76 Cohen MD, Kelly CB and Medaglia AL (2001). Decision Support with Web-Enabled Software. *Interfaces* **31**: 109-129.
- 77 Fourer R and Goux JP (2001). Optimization as an Internet Resource. *Interfaces* **31**, 130-150.
- 78 NEOS Server for Optimization (2001). URL: <http://www-neos.mcs.anl.gov/>
- 79 OSP (2001). URL: <http://www.osp-craft.com>