Documenting Models with the Mathematical Modelling Language LPL

Tony Hürlimann
Department of Informatics
University of Fribourg

October 12, 2004

Abstract

This paper briefly explains the documentation tool in the LPL modelling language. Using two mathematical models as small case studies, the tool is explained in detail. Further motivation for the tool is given. This paper has itself been compiled partially using this tool: the automatically generated documentation of the two case studies is included into the paper (section 5).

Contents

1 Introduction **2**

2 Model Documentation **2**

3 Documentation Implementation **3**

4 Model Documentation in LPL **5**

5 Two small Cases Studies **8**
5.1 Production of Boats (boat) 8
5.2 Producing Robots (robot) 10
5.3 The source code of the two example 13

6 Conclusion **16**

1 Introduction

Mathematical models do not have any meaning. [1] The meaning only comes from our interpretation and application to real-life problems. A symbolic sign in mathematics – such as x – can stand for "the number of aeroplanes of a company", "the price of a product", "the distance of two locations", etc. depending on the context of the application.

Hence, a mathematical model has *many* interpretations and without it, the model only expresses formulas for its own sake without relations to any concrete context. On the other hand, if the interpretation is clear and explicit, then the mathematical formulas – and in general mathematical models, which are just collections of formulas – express the facts we want to formulate in a unsurpassable conciseness and clarity. Its merit is to manipulate them just using the mathematical rules and operations.

If we use mathematical models in a concrete application to solve a practical problem, it is therefore important to enrich it with a clear interpretation. One way to do this is *documentation*: Each symbol and formula must be explained in a phrase, which gives them a meaning. To be useful, the "phrase" should be as short as possible, but should contain all "essential elements" in order to "understand" its meaning in a "specific context". Of course, these criteria are somewhat vague. The "essential elements" and the meaning of "understand" may depend on the addressee, the persons who needs to understand the model. Nevertheless, a mathematical model enriched with natural language text may be a much more useful one than without it.

2 Model Documentation

We can derive from what has been said that documenting a model is not only a desirable part but a necessary one. However, this feature is still badly neglected by the model language designers as well as by the users. Documenting a model takes time and is the least rewarding job, because "the boss" is only interested in the results! The same is true in programming. The only language that supports now documentation extensively is *Java* with its tool, the *javadoc*. There exist tools created a long time ago by [Knuth D. E., 1992] that allow a programmer to generate a complex documentation of the source code, but it seems not to be used widely.

[1] David Hilbert once noted that the content of geometry does not change if we replace the words *point*, *line*, and *plane* by, for example, *chair*, *table*, and *bar*. Wigner in a famous article stated that "mathematics is the science of skilful operations with concepts and rules invented just for this purpose."

However, a well documented model is an invaluable investment (supposing that the model is not used just once). Many models take months to be built – and to document this process makes it much more transparent and accessible to the decision makers. Model documentation can also be an excellent mean to prepare case-studies in learning the modelling building process. Building models is a creative task and must be learnt. One of the best way to learn it, is to study existing models. However, having only the simple mathematical formulation is sometimes not very helpful in understanding the formulas. Hence, a model enriched with explanation is much more useful. In the Internet, virtually thousands of mathematical models – in various language formulations – can be downloaded. How much more useful would they be, if at least an explanation text would accompany them!

Well then, let's code the model here and write a documentation apart from it. Why bother? Why integrate the documentation into the already heavily overloaded code? Because this is the only way to keep the code and the documentation synchronized and consistent. How often the user is willing to write the documentation – at the end of the project, and then the model changes again .... Both – coding and documenting – should be done at the same time and in the same environment. Integrating the documentation into the model itself has another benefit: It can be regenerated (semi-)automatically at any time.

3 Documentation Implementation

Virtually all programming languages allow one to add "comments" to the program code. These comments are normally only useful for the human reader and are entirely left out by the compiler. Hence, they are useless for (semi)-automatical documentation.

Another form to enrich the program (model) code with semantics and meanings is to impose rules on the spelling of identifiers. So, often we hear the requirement that the identifiers should be as "meaningful" as possible. So, for example, of using `CalulateTheAverage` instead of just `Avg` for the name of a function that calculates the average, may help to better understand the code. On the other hand, using `loopIndex` instead of just `i` to denote a loop variable would be certainly an exaggeration. In any case, to attach "too much" meaning to an identifier's name – or symbol, in a mathematical model – conflicts with the requirement to have a concise and short formulation.

A way out of this dilemma is to attach more than one name to the same symbol. So, one can have a long meaningful name for documentation and semantical purposes and a short name just for using it in complex formulas.

It is remarkable that this feature is not used in any – neither programming nor modelling – language, as far as the author is aware. However, it could be useful for various purposes within modelling languages: (1) Mathematical formula, especially indexed ones, tend to be long and unreadable if using long names.

The formula

$$\sum_{i,j,k,h} (p_{i,k} - c_{i,j,k,h}) \cdot X_{i,j,h,k} - \sum_{k,i|i<\#i} s_k \cdot Y_{i,k} + \sum_k (-r_k + p_{\#i,k}) \cdot Y_{\#i,k}$$

would certainly be much more readable than the following one:

$$\begin{aligned}
&\sum_{Periods,ProductionMode,Products,Machines} (Price_{Periods,Products} - \\
&Cost_{Periods,ProductionMode,Products,Machines}) \cdot \\
&QUANTITY_{Periods,ProductionMode,Products,Machines} - \\
&\sum_{p \in Periods,Products|p<\#Periods} Store_{Products} \cdot STORED_{p,Products} - \\
&\sum_{i \in Products} (ResaleValue_i - Price_{\#Periods,i}) \cdot STORED_{\#Periods,i}
\end{aligned}$$

Both represent the same mathematical formula. But the second is so long that it counterbalances the readability.

(2) Multiples names for the same symbol in mathematical models is especially useful for index names. Often we use the same index-set in the same formula for another purpose. As a simple example, we want to sum the rows of a quadratic matrix $A_{i,j}$ with $i, j \in I$. In mathematical notation, we use local indexes i and j :

$$r_i = \sum_{j \in I} A_{i,j} \quad \forall\, i \in I$$

In a modelling language – where the set names are also used as index names such as in LPL, we may code it as:

```
SET I ALIAS i,j; -- this symbol has three names: I, i and j
PARAMETER A{i,j};
          r{i} := SUM{j} A[i,j];
```

In LPL, multiple names can be used for all symbols not just sets.

Still another way to enrich a code with semantics, is to add "qualified" comments. This means that comments added have some syntactical structure in such a way that a program can read and compile them in a well defined manner.

The modelling system LPL allows the modeler to add various documentation parts. They are:

1. Qualified comments for each symbol: A text that explains the meaning of the symbol.
2. Inline documentation: Text added as comments within the formal part of the model.
3. External documentation: Text added beside the formal part of a model.

The three features are explained in the next section.

4 Model Documentation in LPL

The mathematical modelling system LPL includes a LaTeX generation tool. The mathematical and logical formulas are typeset with the free software TeX. Qualified comment and additional text as well as figures and graphics can be included into the formal model specification. From TeX then one can generate a PDF-file, HTML-files, or even XML-files using free software.

As an example, let's take a queen position problem on a chessboard. Suppose the problem is as following: How many queens are needed, and in what position, so that every unoccupied square of the chessboard is under direct attack from some queen? We introduce a binary variable X for each (i, j) position on the chessboard, which is 1 if the position is occupied by some queen, otherwise it is 0. The objective function now is to minimize the number of queens:

$$\min \sum_{i,j} X_{i,j}$$

The constraint of direct attacking can be formulated as following: "For each (i, j) position on the chessboard one must require that it is under attack of at least one queen." "at least" in this context means OR. Hence, one can formulate this in LPL as following (where S is also an index over row/column like i or j):

```
CONSTRAINT
  ATT{i,j}: OR{S|S<>i and S-i+j>=1 and S-i+j<=#S} X[S,S-i+j]
    OR OR{S|S<>i and i+j-S>=1 and i+j-S<=#S} X[S,i+j-S]
    OR OR{S|S<>i} X[S,j] OR OR{S|S<>j} X[i,S];
```

The LaTeX generation tool of LPL translates this constraint automatically into the following LaTeX typeset formula:

$$ATT_{i,j}: \underset{S|S\neq i\wedge S-i+j\geq 1\wedge S-i+j\leq \#S}{\exists} X_{S,S-i+j} \vee \underset{S|S\neq i\wedge i+j-S\geq 1\wedge i+j-S\leq \#S}{\exists} X_{S,i+j-S} \vee \underset{S|S\neq i}{\exists} X_{S,j} \vee \underset{S|S\neq j}{\exists} X_{i,S}$$

However, not only the formal part as such is part of the generated documentation. In LPL, one can add:

(1) **Qualified comments**: For each symbol a text explaining it can be added. For example a variable symbol denoting the quantity of a product could be defined as follows:

```
VARIABLE Quantity ALIAS X  "Quantity of the product in tons";
```

This symbol has two names: `X` and `Quantity`. Both of them can be used alternatively anywhere in the formal part of the model. Besides of this, the symbol has a (short) explanation text:

```
Quantity of the product in tons
```

This text can be much longer and split over several lines.

(2) **Inline documentation**: Any text included in `(**` and `*)` is a comment that will be part of the documentation. (The formal code can also contain ordinary comments, which have the form `(* .... *)`. They are not part of the documentation generated by the documentation tool in LPL.)

(3) **External documentation**: Outside of the formal part – even in another file – one can write extensive documentations, explaining many aspects of the model in a natural language. This documentation is structured in several parts, each beginning with `(** keyword` and ending with `*)`, where `keyword` is a defined word. 10 keywords and their parts are defined as follows:

```
(** intro:    ...   *)
(** problem:  ...   *)
(** modeling: ...   *)
(** explain:  ...   *)
(** solution: ...   *)
(** comm:     ...   *)
(** question: ...   *)
(** answer:   ...   *)
(** step:     ...   *)
(** comm1:    ...   *)
```

The text within the parts (denoted as ...) is copied literally to the LaTeX file. Hence, one should take care to write legal LaTeX code (for example using the character `'$'` will switch the LaTeX compiler to the math mode, therefore if the modeler wants to write a dollar character it must be written as `'\$'`).

The `intro` part introduces the whole model and is placed at the beginning of the output documentation. The `problem` part gives an extensive description of the problem and will follow the `intro` part. Then follows the `modeling` part, which must be written as an enumerated list like the following :

```
(** modeling:
001 First we introduce the variables ....
002 Next we define the data for ....
003 ... etc *)
```

The purpose of the `modeling` part is to give insights into the modelling creation. After this part the documentation tool includes the formal specification of the model, which is followed by the other parts: the `explain` part (which gives further explanation for the model), `solution` part (which adds comments for the solution), `comm` part (which can be added for further comments), the `question` and `answer` part (useful for persons learning the business of mathematical modelling). The `question` and `answer` part must also be in a form of an enumeration :

```
(** question:
1. My first question is ...
2. Second question here ....
3. ... etc *)
(** answer:
1. Answer to the first question ...
2. Answer to the second question ....
3. ... etc *)
```

Several shortcuts can be used in the documentation text:

`@@inc <file>`

to include a file in a verbatim form into the documentation.

`@@lpl`

to include the formal model in verbatim form.

`@ .... @`

to generate verbatim text in the documentation.

```
@@fig file.png(0.8)
```

to include a graphic defined in file "file.pdg".

Further explanation of the LPL documentation tool can be found in the Reference Manual of LPL [Hürlimann T. 2004] and in the tutor examples available within the free distribution of LPL at [Virtual-Optima].

The documentation tool of LPL has been used for large and complex models which generated a documentation of more than 100 pages. Another application is a book – actually in a draft version – containing about 120 case-studies for persons who want to learn mathematical modelling.

5 Two small Cases Studies

The two following examples are implemented using exactly the same mathematical model: They contain two variables (unknown quantities) x and y. The model is:

$$\min\{a \cdot x + b \cdot y) | c \cdot x + d \cdot y \leq e, f \cdot x + g \cdot y \leq h\}$$

where the symbols $a \ldots h$ are given quantities (this is a small linear optimization problem). The two following problems only differ in their interpretations (and the data). The first expresses the production of boats, whereas the second expresses the production of robots. We see clearly, that the models would not have any meaning without the documentation attached to each model.

The next two subsections are automatically generated by the documentation tool of LPL.

5.1 Production of Boats (boat)

Problem

A manufacturer of boats produces two kinds of boats: sailing-boats and motor boats. The profit per sailing-boat (motor boat) is 2400 (2000). Each sailing boat needs 4 m^2 sailing cloth and the availability is 400 m^2. Each motor boat needs a motor, and their availability is 120 pieces. Furthermore, a sailing-boat needs 8 m^2 fibreglass and a motor boat needs 4 m^2. 1000 m^2 of fibreglass is available at the moment. How many boats of each type can the manufacturer produce in order to make a maximal benefit?

Modelling Steps:

1. The unknowns are the number of sailing-boats (motor boats). We denote them with x and y.

2. To produce x sailing boats, we need $4x$ m^2 sailing cloth. We have 400 m^2 maximally. Hence $4x \leq 400$.

3. To produce y motor boats, we needs y motors. We have 120. Hence, $y \leq 120$.
4. To produce x sailing boats and y motor boots, we needs $8x+4y$ m^2 fibreglass. We have 1000 m^2. Hence, $8x + 4y \leq 1000$.
5. The benefit of x sailing boats and y motor boots is: $2400x + 2000y$ (in).
6. This quantity we like to maximize.

The model

VARIABLE

x

Number of sailing-boats to produce

y

Number of motor boats to produce

CONSTRAINT

$NeedCloth : 4 \cdot x \leq 500$ (1)

Need and availability of sailing cloth

We suppose here, that a sailing-boat has no motor

$NeedMotor : y \leq 120$ (2)

Need and availability of motors

$NeedFibre : 8 \cdot x + 4 \cdot y \leq 1000$ (3)

Need of fibreglass

MAXIMIZE $2400 \cdot x + 2000 \cdot y$

WRITE

$benefit, x, y$ (4)

END

Solution

Solving the problem reveals that the manufacturer can produce 65 sailing-boats and 120 motor boats. The benefit is 396000.

Question 1 *(Answer to 1)*

1. *Could the manufacturer produce more sailing-boats, if he could buy more sailing cloth at the moment?*

Answer 1 *(Question of 1)*

1. *No! The need of fibreglass is the limiting constraint, not the sailing cloth.*

5.2 Producing Robots (robot)

Problem

A firm produces two type of robots called 'Marie' and 'Jules'. Three production steps must be carried out:

1. Production of the components, which takes 5 hours for each robot Marie and Jules, with a total capacity of 350 hours per week,
2. Mounting (capacity = 480) taking 4 hours for Marie and 8 hours for Jules,
3. Testing (capacity = 300) taking 6 hours for Marie and 2 hours for Jules.

There are already 20 robots of type Marie and 15 Jules ordered. The profit for Marie is 300 and for Jules 200. How many robots of each type can be produced per week, if the firm wants to maximize the selling profit?

Modelling Steps:

1. We introduce two variables: x and y for the quantity (per week) of the two types of robots to produce.
2. The components-step has a capacity of 350 hours per week. A robot Marie takes 5 hours, a robot Jules also takes 5 hours. Therefore: $5x + 5y \leq 350$.
3. The mounting-step has a capacity of 480 hours per week. A robot Marie takes 4 hours, a robot Jules takes 8 hours. Therefore: $4x + 8y \leq 480$.
4. The testing-step has a capacity of 300 hours per week. A robot Marie takes 6 hours, a robot Jules takes 2 hours. Therefore: $6x + 2y \leq 300$.
5. At least 20 of type Marie and 15 of type Jules must be produced, hence: $x \geq 20, x \geq 15$.
6. Maximizing the profit means to maximize: $300x + 200y$.

The model

VARIABLE

x

Number of robots of type 1

y

Number of robots of type 2

CONSTRAINT

$$Components : 5 \cdot x + 5 \cdot y \leq 350 \quad (1)$$

Capacity of the components

$$Mounting : 4 \cdot x + 8 \cdot y \leq 480 \quad (2)$$

Capacity of the mounting step

$$Testing : 6 \cdot x + 2 \cdot y \leq 300 \tag{3}$$

Testing capacity

$$Order1 : x \geq 20 \tag{4}$$

Already order of type 1

$$Order2 : y \geq 15 \tag{5}$$

Already order of type 2

MAXIMIZE $300 \cdot x + 200 \cdot y$

END

Solution

The optimal solution is $(x, y) = (40, 30)$, which can be verified by Figure 1.

Comments

In a graphical way, the model represents the solution space as shown in Figure 1. That is, every (x, y)-point within the gray polygon represents a possible (feasible) production that fulfills all constraints. All other points in the two-dimensional space violate at least one of the constraints.

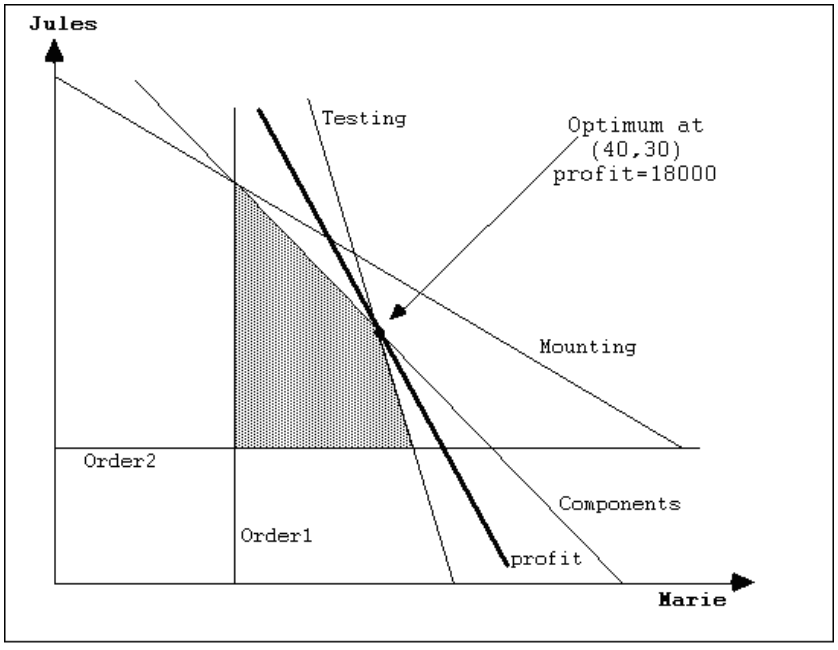

Figure 1: The Feasible Space

This mathematical model can be coded in LPL as follows:

```
MODEL Robots "Producing Robots";
  VARIABLE  x "Number of robots of type 1";
            y "Number of robots of type 2";
  CONSTRAINT
    Components: 5*x + 5*y  <=  350 "Capacity of the components";
    Mounting:   4*x + 8*y  <=  480 "Capacity of the mounting step";
    Testing:    6*x + 2*y  <=  300 "Testing capacity";
    Order1:       x        >=   20 "Already order of type 1";
    Order2:              y  >=   15 "Already order of type 2";
  MAXIMIZE profit: 300*x + 200*y;
END
```

1. The model begins with a `MODEL` keyword and ends with `END`.
2. It consists of a sequence of entity declarations.
3. Each entity begins with an keyword and ends with a semicolon.
4. There are four kinds of entities here: `MODEL, VARIABLE, CONSTRAINT`, and `MAXIMIZE`. There is no need to repeat them for consecutive entities of the same type
5. `VARIABLE` introduces a list of variables.
6. `CONSTRAINT` introduces the model constraints.
7. The objective function, called 'profit' begins with `MAXIMIZE`.

Question 2 *(Answer to 2)*

1. *Verify the solution of this model by running it.*
2. *What happens of the* `mounting` *capacity is extended to 1000 hours?*

Answer 2 *(Question of 2)*

1. *Open lplw.exe in the file browser. Choose Menu "File/Open" and open robot.lpl. Then choose Menu "Run/Run Model". Click tab 'TABLE', and then click 'x' in the Tree (left part in the LPL application). The value 40 is displayed. Now click on 'y', the value 30 is displayed.*
2. *Nothing happens, as can be verified easily by the Figure 1. The mounting is not a limiting production step.*

5.3 The source code of the two example

The source code of the boat model from which the documentation was generated, is as follows:

```
MODEL Boat "Production of Boats";
  VARIABLE x "Number of sailing-boats to produce";
           y "Number of motor boats to produce";
  CONSTRAINT
    NeedCloth: 4*x <= 500 "Need and availability of sailing cloths";
    (** We suppose here, that a sailing-boats has no motor *)
    NeedMotor: y <= 120  "Need and availability of motors";
    NeedFibre: 8*x + 4*y <= 1000  "Need of fibreglass";
  MAXIMIZE benefit: 2400*x + 2000*y;
  WRITE benefit, x,y;
END

(** problem:
A manufacturer of boats produces two kinds of boats:
sailing-boats and motor boats. The profit per sailing-boat (motor
boat) is 2400 (2000). Each sailing boat needs 4 $m^2$ sailing
cloth and the availability is 400 $m^2$. Each motor boat needs a
motor, and their availability is 120 pieces. Furthermore, a
sailing-boat needs 8 $m^2$ fibreglass and a motor boat needs 4
$m^2$. 1000 $m^2$ of fibreglass is available at the moment. How
many boats of each type can the manufacturer produce in order to
make a maximal benefit? *)

(** modeling:
001 The unknowns are the number of sailing-boats
(motor boats). We denote them with $x$ and $y$. 002 To produce $x$
sailing boats, we need $4x$ sailing cloth. We have 400 maximally.
Hence $4x \leq 400$. %
003 To produce $y$ motor boats, we needs $y$ motors. We have 120.
Hence, $y \leq 120$. %
004 To produce $x$ sailing boats and $y$ motor boots, we needs
$8x+4y$ fibreglass. We have 1000. Hence, $8x+4y \leq 1000$. %
005 The benefit of $x$ sailing boats and $y$ motor boots is:
$2400x+2000y$. %
006 This quantity we like to maximize. *)

(** solution:
Solving the problem reveals that the manufacturer
can produce 65 sailing-boats and 120 motor boats. The benefit is
```

```
396000 (). *)

(** question:
 1. Could the manufacturer produce more
sailing-boats, if he could buy more sailing cloth at the moment?
*)

(** answer:
1. No! The need of fibreglass is the limiting
constraint, not the sailing cloth. *)
```

The source code of the model robot is as follows:

```
MODEL Robots "Producing Robots";
  VARIABLE  x "Number of robots of type 1";
            y "Number of robots of type 2";
  CONSTRAINT
    Components: 5*x + 5*y  <=  350 "Capacity of the components";
    Mounting:   4*x + 8*y  <=  480 "Capacity of the mounting step";
    Testing:    6*x + 2*y  <=  300 "Testing capacity";
    Order1:       x        >=   20 "Already order of type 1";
    Order2:              y  >=   15 "Already order of type 2";
  MAXIMIZE benefit: 300*x + 200*y;
END

(** problem:
A firm produces two type of robots called 'Marie' and
'Jules'. Three production steps must be carried out:
\begin{enumerate}
\item Production of the components, which takes 5 hours for each
robot Marie and Jules, with a total capacity of 350 hours per
week, \item Mounting (capacity = 480) taking 4 hours for Marie and
8 hours for Jules, \item Testing (capacity = 300) taking 6 hours
for Marie and 2 hours for Jules.
\end{enumerate}
There are already 20 robots of type Marie and 15 Jules ordered.
The profit for Marie is 300 and for Jules 200. How many robots
of each type can be produced per week, if the firm wants to
maximize the selling profit? *)

(** modeling: %
001 We introduce two variables: $x$ and $y$ for the
```

```
quantity (per week) of the two types of robots to produce. %
002 The components-step has a capacity of 350 hours per week. A
robot Marie takes 5 hours, a robot Jules also takes 5 hours.
Therefore: $5x+5y \leq 350$. %
003 The mounting-step has a capacity of 480 hours per week. A
robot Marie takes 4 hours, a robot Jules takes 8
hours. Therefore: $4x+8y \leq 480$. %
004 The testing-step has a capacity of 300 hours per week. A robot
Marie takes 6 hours, a robot Jules takes 2 hours.
Therefore: $6x+2y \leq 300$. %
005 At least 20 of type Marie and 15 of type Jules must be
produced, hence: $x\geq20, x\geq 15$. %
006 Maximizing the profit means to maximize: $300x+200y$. *)

(** comm: %
In a graphical way, the model represents the solution space as
shown in Figure \ref{robot01}. That is, every $(x,y)$-point within
the gray polygon represents a possible (feasible) production that
fulfills all constraints. All other points in the two-dimensional
space violate at least one of the constraints.

@@fig robot.png(0.8) "The Feasible Space"

This mathematical model can be coded in LPL as follows:

@@lpl *)

(** step: %
1. The model begins with a @MODEL@ keyword and ends with @END@. %
2. It consists of a sequence of entity declarations. %
3. Each entity begins with an keyword and ends with a semicolon.
4. There are four kinds of entities here:
   @MODEL, VARIABLE, CONSTRAINT, and MAXIMIZE@. There is no need to
   repeat them for consecutive entities of the same type
5. @VARIABLE@ introduces a list of variables. %
6. @CONSTRAINT@ introduces the model constraints. %
7. The objective function, called 'profit' begins with @MAXIMIZE@.
*)

(** solution:
The optimal solution is $(x,y)=(40,30)$, which can
be verified by Figure \ref{robot01}. *)
```

```
(** question:
1. Verify the solution of this model by running it. %
2. What happens of the @mounting@ capacity is extended to 1000
hours? *)

(** answer:
1. Open lplw.exe in the file browser. Choose Menu
"File/Open" and open robot.lpl. Then choose Menu "Run/Run Model".
Click tab 'TABLE', and then click 'x' in the Tree (left part in
the LPL application). The value 40 is displayed. Now click on 'y',
the value 30 is displayed. %
2. Nothing happens, as can be verified easily by the Figure
\ref{figrobot}. The mounting is not a limiting production step. *)
```

This ends the source code of the two models.

6 Conclusion

Documenting models is extremely important. It is surprising that this has been overlooked by most modelling language designers, as well as by the programming language designers. They are two notable exceptions: Java with its Javadoc tool and the Literate Programming tools implemented by Donald E. Knuth to document his (large) programs. The whole TEX system has been documented with this feature.

Documenting models is even more important than for programs, since a mathematical model in itself has no meaning, and only its connection to a real-life problem (its interpretation) fills it with life.

We are aware that the documentation tool in the modelling language LPL is not the final result, but only a first step in documenting models. Many questions are open: (1) What about result documentation, that is, documentation directly generated by a solution or by many solution variants of the model. This realm is *model reporting*, but it should be linked in a way to the overall documentation. (2) What about structural documentation of the formal model, generated directly from the syntax tree of a model. Here many graphical "views" to the models could be generated and included to a documentation. (3) Last but not least, what about "two-way" documentation? A report as a part of result documentation, for example, is a "one-way" documentation, if printed to a paper. But if it is "printed" to a screen, it could be used also as an "input mask" for the data, hence as editing tool.

References

[Hürlimann T. 2004] HÜRLIMANN T., (2004), Reference Manual for the LPL Modelling Language, Working Paper, Version 4.50, August 2004, Departement of Informatics, University of Fribourg, (see www-virtual-optima.com).

[Hürlimann T. 1999] Hürlimann T., (1999), Mathematical Modeling and Optimization, An Essay for the Design of Computer-Based Modeling Tools, Kluwer Academic Publ., (Applied Optimization 31).

[Knuth D. E., 1992] Knuth D. E., (1992), Literate Programming, CSLI (Center for the Study of Language and Information), Lecture Notes Number 27.

[Virtual-Optima] VO, `www.virtual-optima.com` , The home page of Virtual-Optima.