# Models crossing the boundaries of tools

Magne Myrtveit
ModellData AS
N-5120 Manger
Norway
Phone: +47 5637 4009
Fax: +47 5637 3500
Internet: magmy@modeld.no

(ABSTRACT)

As the field of system dynamics matures, and the number of available tools increases, the problems related to incompatible model formats consume increasingly large portions of researchers', educators', and model users' time. Manifestations of these problems are many: Researchers may be unable to communicate models because they are using different tools (or different versions of the same tool). Educators face problems distributing models because schools use different tools. Authors have problems including a models disk with their books, as readers use different tools.

Similar problems have been experienced with other tools, like word processors and drawing programs. The kind of solutions that are used for transferring documents between word processors can be applied also to system dynamics models. One solution would be to implement a series of model conversion filters in each SD tool. (This corresponds to the way MS-Word may load and store files in WordStar, Word Perfect, Works, and Write format.) Another solution would be to define a common standard file format for interchanging models. (This corresponds to formats like plain text and Rich Text Format for documents, and BMP, EPS, GIF, PCX, TIFF, and WMF for graphics.)

The latter approach has several advantages. 1) Each software tool need only support one foreign file format, in addition to its native file format. 2) The software tools need no knowledge of each others native file formats.

The benefits of developing a model interchange file format are many; 1) Models may be stored in a common format (e.g., in libraries). 2) Models may be transferred in a common format (e-mail, diskettes included in text books, etc.). 3) Co-development of models. 4) Easier use of several software tools by the same user.

## The problem

For more than 35 years work has been done within the field of system dynamics. Alongside the development of the field, computer hardware and software, existing tools are revised and new tools emerge. The inclusion of graphical editing capabilities and new language constructs brings new power and diversity to the set of tools.

In parallel with the development of tools, the file formats for storing data have been revised, extended and often totally redefined.

The incompatibilities in file formats have caused enormous amounts of extra work in moving files between hardware platforms, operating systems and software tools. Talking to users of system dynamics tools, it is striking how often they refer to problems related to incompatible file formats for models. For educators problems may relate to the development of model libraries that can be used for any of the most popular tools. Authors and publishers may want to include models with text books. Researchers using different tools may want to exchange models or participate in co-development of models. Advanced users may want to use more than one tool and at the same time be able to transfer models between tools. In parallel with the growing use of Internet many users definitely want to transfer models using e-mail.

## Solutions from other software families

Development paths similar to the SD tools take place within other software families. The different word processing tools have undergone major changes since the time of WordStar; spread sheets of today have capabilities far beyond Supercalc; and modern painting programs offer much more than the first version of Mac Paint.

To overcome some of the problems, commercial converter programs have been developed, especially for translation between different bitmap formats (PCC, PCX, CUT, BMP, GIF, IFF, TIFF, etc.). At some point tools started to include import and export filters for loading and storing files in foreign formats. This has now become the standard for most tools within the largest software families (word processing, spread sheets, databases, illustration software, etc.). For example, as a user of MS-Word you can load files produced in WordStar, Word Perfect, Works, and Write.

## Suggested solution

Basically there are three solutions to the problem of interchanging models in a way that is independent of tool. The first solution is manual entry of a model into each tool that is going to be supported. The second, is to include file converters for the most popular file formats in each of the software packages. A third solution is to define a common model interchange format that would be supported by all tools.

The solution with import/export filters for files created with other tools have two main disadvantages. First of all, the number of conversion filters that need to be made will increase rapidly as the number of tools (and versions of file formats) grow. If there are N different file formats, each tool has to implement N-1 import and export filters. The total number of files that must be made for all tools becomes $(N^2-N)/2$, which is a rapidly growing number for increasing values of N (see Figure 1). Another problem with this solution is that all file formats must be

made available to the other software vendors in order to make the import/export filters. This would require large overhead in terms of documentation and administration, and might also create other difficulties.

The third alternative has the advantage that each tool needs to make only one import/export filter in addition to its native file format (see Figure 2). Managing one common file format would be less time consuming than exchanging information about all formats and new revisions.

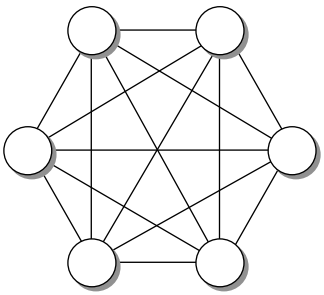


**Figure 1: All-to-all file conversion**

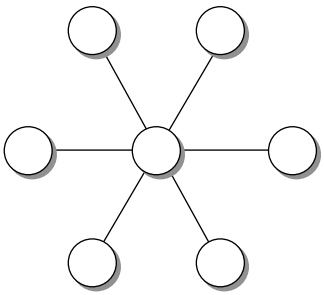


**Figure 2: Common interchange format**

Based on these observations, it is clear that a common interchange format for simulation models would benefit both software developers (reduced costs) and users (one preferred format for interchanging models).

## Requirements

The model interchange format (MIF) should be a system independent standard for storing and transferring simulation models. The standard should at least meet the following requirements:

R1: A large family of simulation models should be possible to store in the MIF format.
R2: A model exported in the MIF format should be possible to import without any loss of information when imported into the same software.
R3: Models should transfer easily between operating systems and via e-mail.
R4: The standard should be designed with the possibility for future expansions in mind.
R5: Systems should be allowed to support only parts of the standard.
R6: It should be possible to include models in MIF format in clipboard transfers, allowing models to be copied between SD tools using Copy and Paste.

## Design guidelines

The above list of requirements leads to some high level guidelines in designing and using the standard.

R1 may be achieved by choosing a layered standard, where the lowest level encompasses the majority of existing simulation models.

R2 may be achieved by allowing each software to include native information (information that may be skipped by the other software packages) in the exported MIF files.

R3 may be achieved by using 7-bits ASCII text files to store models in MIF format, and selecting a unique way to represent characters outside the ASCII character set (escape sequences).

R4 may be achieved by labeling native information in a way that will never be in conflict with future expansions of the standard. Using a "tagged" file format (where each piece of information is associated with a label) makes R4 easier to fulfill.

R5 may be fulfilled by defining a layered standard, and/or by defining the standard as a collection of features from which a given model (or software) may use (support) a subset.

## Suggested definition of the standard

In addition to the requirements and guidelines listed above, we must answer one important question before defining the standard:

*What is a dynamic model?*

Is it a set of equations? What about accumulator–flow diagrams? How do we store time graphs and time tables. What about pictures, text and frames that may be part of a graphical interface?

Traditionally, models are looked upon as a set of equations. But this way of representing a model makes it difficult to include all the extra information that is part of models created in modern SD tools. (As an example, it does not seem natural to describe a time graph as an equation.)

Maybe the most versatile description of a model would be that *a model is a collection of objects*, where some objects are variables (defined by equations) and others display information or communicate with the user or the operating environment (files, applications, hardware).

A MIF file may contain the following objects:

Model Objects (variables)
Static Objects (pictures, texts, frames, ...)
Dynamic Objects (time graphs, time tables, bars, ...)

The above objects may make references to:

Functions
Ranges or FOR variables (used to define arrays)
Co-models
Macros

In addition, a MIF file may include information like the following:

Originator (name of software package and version)
Top level information about model (documentation, statistics, ...)
"MIF-Level" of the contained model

### *An approach based on equations*

It is possible to extend the syntax used to describe a model as a set of equations. The POWERSIM® equation:

```
aux Births = Population * BirthRate
```

defines an auxiliary variable named *Births* equal to the expression *Population* * *BirthRate*. The corresponding Professional DYNAMO™ equation is:

```
R BIRTHS.KL = POP.K * BIRTHRT
```

The above equations may be looked upon as examples of definitions based on the following template:

```
<tag> <name> = <value>
```

This means that the equation languages can be interpreted as tagged formats according to the guidelines listed earlier. For objects with several attributes, it would be possible to define a unique tag for each attribute. As an example, a variable with definition, documentation, and graphical symbol could be described using three tagged expressions, like this:

```
aux Births = Population * BirthRate
doc Births = "Number of new-borns per time unit"
pos Births = (130, 102)
```

A disadvantage with this approach is that it is unstructured. The number of tags will grow rapidly and the information concerning one object can be scattered throughout the file. Another problem is that all objects must be named, which need not always be the case (for example with pictures in a diagram).

### *An approach based on objects*

Instead of basing the standard on equations, we can focus on objects and object attributes. Successful file formats like TIFF (tag image file format), IFF (interchange file format) and RTF (rich text format) are all based on the grouping of tagged information in a hierarchical file structure. The RTF format (Microsoft 1994) has the additional feature that it is text based (see R3). Many of the objects and attributes of a simulation model (e.g. pictures, name of author, summary info) are already defined by the RTF standard.

### *Formal MIF syntax*

Based on the RTF standard we may define an object as a RTF "group", which is a collection of data enclosed in braces ({}). Objects may be nested. Attributes may be stored as RTF "control words", which takes one of the following forms:

\LetterSequence<delimiter>
\LetterSequence<number><delimiter>

Control words start with a backslash (\). The LetterSequence is made up of lowercase alphabetic characters in the range a-z. A number is a decimal integer, composed from the digits 0-9 and may have a leading minus sign. (In the MIF standard we may want to include decimal numbers.)

In the following, the MIF syntax is described using a syntax based on the Backus-Naur Form:

| Syntax | Meaning |
|---|---|
| ‘c’ | A literal |
| <text> | A nonterminal. |
| a | The terminal control word a, without a parameter. |
| aN | The terminal control word a, with a parameter N. |
| a? | Item a is optional. |
| a+ | One or more repetitions of item a. |
| a* | Zero or more repetitions of item a. |
| a b | Item a followed by item b. |
| a \| b | Item a or item b. |
| a & b | Item a and/or item b, in any order. |

### *Contents of MIF file*

At the top level a MIF file can be defined like this:

<file> ‘{‘ <header> <model> ‘}’

### *MIF file header*

The header contains global information about the file, including file version. The following, at least, should be contained in the header:

<header> **\mifN** <charset> <originator>

The numeric argument to the **mif** control word defines the version of the MIF file. The first version will have the control word **\mif0**. Charset holds the character set used for the rest of the file. Valid control words include **\ansi** and **\mac** for the standard Windows and Macintosh character sets, respectively. Originator should identify the software that created the file, like this:

<originator> ‘{‘ **\origN** <text> ‘}’

Text holds the name of the software and the numeric argument to orig the version number times 100. As an example, POWERSIM version 2.01 would be stored as:

```
{\orig201 POWERSIM}
```

### *MIF model area*

A model will have some global information in addition to the set of objects from which it is composed. Following the RTF standard, we define a model like this:

<model> <info>? <modfmt>* <object>*

### *The information group*

This group contains title, subject, author, etc. The definition can be taken directly from the RTF standard. An example of the information group follows:

```
{\info{\title The coffee cup}{\author Arne-Helge Byrknes}}
```

*Model format group*
This group will hold global options for the document, including editing options, viewing options and page information.

*Object groups*
Variables, time graphs, bars, pictures, etc. will be defined as objects. Below are some examples of how objects can be stored.

*Variable objects*
A variable object may be defined like this:

| | |
|---|---|
| <var> | '{' **\var** <name> <doc>? <def>? <pos>? <size>? '}' |
| <name> | '{' **\name** <ident> '}' |
| <doc> | '{' **\doc** <text> '}' |
| <def> | <expression> |
| <pos> | '{' **\pos** <x> <y> } |
| <size> | '{' **\size** <x> <y> } |
| <x> | '{' **\xN** '}' |
| <y> | '{' **\yN** '}' |

The name group defines the name of the variable. Doc defines the documentation (comment) associated with the variable. Def represents the right hand side of the variable definition. Pos and size defines the position and size of the graphical representation of the variable in the display. Measurements are in twips (1/1440 of an inch, which is the same as 1/20 of a point). Below is an example:

```
{\var{\name Births}{\doc Number of new-borns per time
unit}{\def Population * BirthRate}{\pos\x130\y102}{\size
\x720\y720}}
```

*Time graph objects*
As another example, a time graph can be defined like this:

| | |
|---|---|
| <tgraph> | '{' **\tgraph** <doc>? <varlist> <pos>? <size>? <series>* '}' |
| <varlist> | '{' **\varlist** ( <ident> ';' ) * '}' |
| <series> | '{' **\series** <name> <timespec>? ( <number> ';' )* '}' |
| <timespec> | '{' **\times** <from> <to> <step> '}' |
| <from> | **\fromN** |
| <to> | **\toN** |
| <step> | **\stepN** |

Doc will hold the documentation associated with the time graph, if any. Varlist will define the parameters to the graph (dependent variables). Series holds data present in the time graph. An example is displayed below:

```
{\tgraph{\doc Development of the population over
```

```
time}{\pos\x150\y3040}{\size \x2000\y3000}{\varlist
Population;}{\series{\times\from0\to100\step1}2;3.4;4;4.12;4.7
6;}}
```

Information about scaling and labels of axis, colors etc. should be added to the definition.

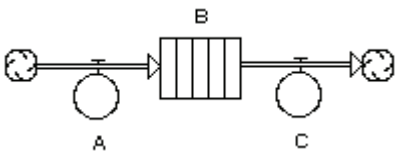
Note that the above examples are just illustrations indicating how a possible standard can be defined.

## Defining levels of the standard

As the different software packages have widely different features that do not necessarily lie along a linear path of natural progression, it may be difficult to stick to a strictly hierarchical standard. Looking at the features of the different tools, there seem to be several dimensions to move along:

| Feature | Significance |
|---|---|
| *variable types* | Some tools have more variable types than others. Examples include queues, conveyors, ovens. |
| *scalar functions* | The number of functions supported varies. The meaning of some functions with equal names varies between tools. |
| *array variables* | Some tools support arrays, others don't. |
| *array functions* | The set of functions operating on arrays varies between tools. |
| *range definitions* | The way ranges (FOR variables) are defined varies. |
| *diagram editing* | Some tools do not have diagram editing capabilities. |
| *links* | Some tools have more types of links than others. Examples include initialization links and delayed links. |
| *flows* | Some tools have more types of flows than others. An example is unidirectional flows. |
| *snapshots/aliases/ghosts* | Are they supported by all tools with diagram editing capabilities? |
| *dynamic objects* | The set of available objects vary. Most tools support time graph, time table, and scatter graph. |
| *static objects* | The set of available objects vary. Most tools support text, frame and picture. |
| *macros* | Only a few tools support macros. |
| *implementation limits* | The maximum number of variables, length of variable names, etc. vary. |

For some of the features that vary between tools, alternative features may be used in some or all of the tools. As an example, a conveyor in STELLA/ithink® can be represented as an auxiliary defined by a DELAYPPL function in POWERSIM.



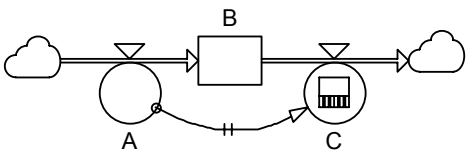



**Figure 3: Conveyor in STELLA/ithink** **Figure 4: Corresponding POWERSIM diagram**

When compiling libraries of models to be included in text books or in databases for public use, the features used by the models should be specified. One library of models may, for example, contain models that do not use arrays. Another may contain models that do not use diagram symbols, etc.

## Resolving incompatibilities

### *Equation based models without diagram information*

Importing models without diagram information will be difficult to implement in systems like STELLA/ithink and POWERSIM, as these packages do not have a text mode for editing models.

### *Macros*

Macros should be expanded during export, since not all systems support macros. The macros can still be part of the exported file if alternative definitions are produced (for example using native control words).

### *Diagrams*

The diagram "languages" are not 100% compatible. As an example, POWERSIM has shared rates, delayed links and initialization links. As another example, STELLA/ithink has queues, conveyors, ovens, and unidirectional flows.

Differences like these can be resolved by converting unsupported features in a source model into a compatible model fragments in the destination model. As an example, shared rates (Figure 5) can be implemented in STELLA/ithink as two rates where one rate is defined equal to the other (Figure 6).

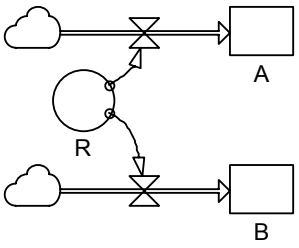



**Figure 5: Shared rate in POWERSIM**

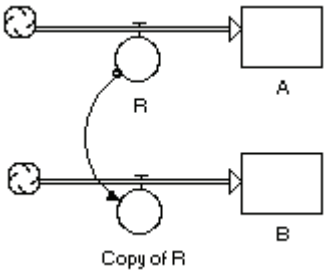



**Figure 6: Equal rates in STELLA/ithink**

### *Functions*

The lists of functions vary. The functions of the MIF format must be precisely defined, and the necessary mappings done by the export/import filters of each package. Some functions may be impossible to express in some of the packages, and should therefore be left out from the lowest level of the standard. (A mapping from function to equivalent structure may be generated during import.)

*Variable types*

All current SD modeling tools support levels and non-levels. All other variable types can in principle be mapped to levels or non-levels (except for queues, ovens, conveyors which require larger structures to be implemented). The lowest level of the standard should mention a list of common variable types, and the appropriate mapping of each variable type to one of the two basic variable types.

As a final note on variable types, it is possible to operate with only one basic variable type. This can be done by introducing the INTEGRATE function and use that to define levels. As an example, a level L with initial value L0 and flow F can be defined like this:

```
L = L0 + INTEGRATE(F)
```

## Further work

One way to continue work on the standard would be to establish a group of software developers, users and representatives from the system dynamics society. A project in this category would probably qualify for support from national or international standards organizations. In addition to the standard model interchange format, a number of other aspects can be made subject to a standardization (does a *red* link mean *same* or *opposite* influence?). As the fields of system dynamics and systems thinking mature, there will be a growing need for a permanent standardization committee. Ultimately, ANSI or ISO standardization can serve as solid and stable grounds for continuos and synergetic growth within the field.

## References


Microsoft Corporation 1994. *Rich Text Format (RTF) Specification*

ModellData AS 1994. *POWERSIM 2.0 User's Guide and Reference*


## Trade marks