

Improving the model development cycle by automatic configuration of modelling tools

K. de Jong^a, C.G. Wesseling^b, D. Karssenberg^a

^aDepartment of Physical Geography, Utrecht University,
Heidelberglaan 2, PO box 80115, Utrecht, The Netherlands, (k.dejong@geog.uu.nl)

^bPCRaster Environmental Software,
PO box 427, Utrecht, The Netherlands

Abstract: Model development is a difficult and timely process involving various steps to create, check and improve the model. Typically all kinds of software tools are used to facilitate this process. Central to model development is the model definition and we show that, apart from calculating model results, it can be used to tailor some of the tools used to the model at hand. This speeds up model development and might lead to better models.

Keywords: environmental modelling; model development cycle; model definition

1 INTRODUCTION

Model development is often a difficult and time consuming part of environmental modelling. During development knowledge about the environmental processes is translated into model statements. This requires both field experience and modelling skills and only the most trivial models are created correct the first time. In practise model development is an iterative process during which the model is defined, tested and redefined again. Only after a number of iterations does the development process result in a model which adheres to the developers criteria. And even then there are various reasons why in a later stadium this 'finished' model has to be redefined again, for example because better or other input data has become available, further knowledge about the physical processes at play has been acquired, other output data is required, different model structures for the same problem need to be compared [see for example Van der Perk, 1997], or when the model is applied in another area.

Clearly it is an advantage if the modelling environment supports the creation of models in such a way that they can be easily tested and adjusted. An important aspect of the modelling environment is the set of tools available to define the model, visualise and analyse the data, debug and test the model. Since these tools are used repeatedly they should

perform well and enable the developer to do his or her work in the best possible way. A second important aspect is the language which is used to implement the model. The advantages of high level Environmental Modelling Languages (EML) compared to system programming languages like Fortran and C++ are described in Karssenberg [2002] for the case of distributed hydrological model development. It is concluded there that the main benefits are: better re-usability of program code (compared to re-implementing generic algorithms in a system programming language), lack of technical details in the model definition, short development time and easier to learn. In general, when constructing continuously evolving models and if tailoring to study aims and field data is important, then a modelling environment with an EML is better suited than one with a system programming language. In this article it is assumed that an EML is used to implement the model. Examples of EMLs are STELLA [STELLA, 2002], ARC Macro Language (AML) [AML, 2002] and PCRaster [Wesseling et al., 1996; PCRaster, 2002]. In the examples presented the PCRaster EML is used. PCRaster is an environmental modelling environment developed by the Department of Physical Geography of Utrecht University and PCRaster Environmental Software.

In the next sections we first describe the practise of model development including the typical modelling

tools used. After that we propose a way to automatically tailor some of the modelling tools to the needs of the specific model created, in order to optimise the model development cycle. Examples will demonstrate how these techniques can be used.

2 MODEL DEVELOPMENT CYCLE

An ideal and complete model development cycle is shown in Figure 1. In practise, not every cycle will consider all the phases shown: this depends on the model characteristics, available field data and other resources. We assume that the required input data are already gathered and ready to be read by the model. Data collection and collation are not considered part of the model development process. The iterative process of model development starts with (re)defining the structure of the model. This phase comprises analysis of model requirements, analysis of input data, literature study, translation of model concepts to model statements and model documentation. The tools used in this phase are: model script editor, data visualisation tool, Exploratory Data Analysis (EDA) tool and a statistical tool. Apart from the editor these tools are used to analyse the input data sets and generate hypotheses about how they are related to the output data sets. Next, the model is executed by the model engine. In our case this tool is an interpreter which reads and checks the model script and calculates the output data. In case a system language is used to write the model the tools used in this phase would be the compiler used to create an executable model and the executable itself. When syntactical and semantical errors are found the script has to be adjusted. After the model results are calculated they are analysed by using the visualisation, EDA and statistical tools. There are several reasons why the model has to be adjusted before continuing, for example because the output data values are out of range or the temporal resolution is wrong. When the model generates output as expected the value of the calibration parameters (if present) need to be determined. A calibration tool like PEST [PEST, 2002] can be used in this phase. Again, it can be decided that the model has to be adjusted. The most important reason for this is that the model can't be calibrated to field data at all. After the model has been calibrated a validation tool is used to validate the model results with a field data set. When the results differ too much from the field data, the model must be adjusted once again. In general, redefining the model structure continues until it satisfies the criteria of the study. And when these criteria change often the structure has to change too.

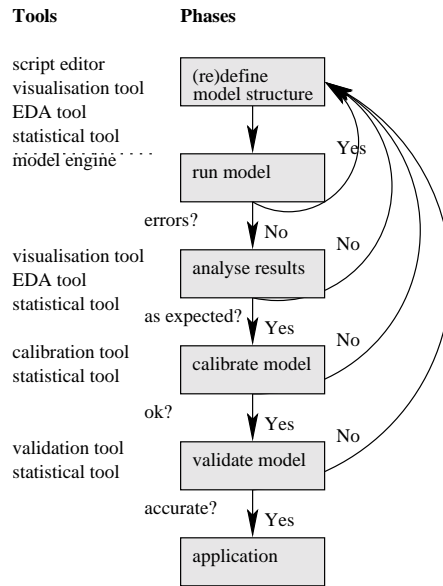


Figure 1: Model development cycle with tools used.

Only after validation proved that the model is accurate enough in calculating output, it can be used as a standard modelling tool by users who are not necessarily familiar with model building, for example most decision makers. Therefore an easy to use graphical interface to the model might be needed. During model application the model structure remains fixed. For reasons already mentioned in the Introduction it can be decided that the model must be redefined in a later stadium by the developer.

3 PROBLEM AND IMPROVEMENTS

It should be clear by now that in general the model is redefined often during the model development cycle and that various tools are used repeatedly. Since the tools are used so often even small improvements in their functionality can make the whole model development process more efficient.

A major disadvantage of most general modelling tools like visualisation and statistical tools is that every time they are used they need to be configured. If, for example, we want to visualise the data sets generated by the model, we have to explicitly 'tell' the visualisation tool to do so each time we want to see the data. The problem in this case is that the visualisation tool can't figure out which data sets are generated by the model. We might be able to automate such often used tool configurations by writing batch files, but these have the drawback that each time the model is redefined, often they have to be adjusted too. The approach described in the

next sections comes down to automatically converting the model script and adjusting the tools in such a way that all model characteristics are readily available for the modelling tools to use.

4 MODEL DEFINITION

The model definition is a formal description of the model concepts. This can be in the form of a model script or a set of rules and formula's on paper. In case of a model script the Environmental Modelling Language (EML) is used to write the model definition. It is designed to be as easy and powerful as possible for the modeller. This results in a language which is terse and often the models written in it contain implicit model characteristics. Examples of these are the data types (e.g.: raster map and time series) and value types (e.g.: boolean and scalar) of the datasets used and the spatial resolution and extent. In the next example the `slope` function is used to calculate the gradient of the digital elevation model `b`.

```
a = slope(b);
```

Only the modeller and/or the model engine 'know' that `b` and `a` are both spatial data sets with scalar value type and that as cells contain percentages. This implicit information can be deduced (made more explicit) from the definition but external information which is not present in the model script is required (about the function prototypes for example). The following example shows how the previous script fragment looks when made more explicit. We use an application of the extensible markup language (XML) [XML, 2002] in the examples. This technique is suitable for passing information between applications.

```
<statement>
  <output type="spatial" valuescale="scalar"
    units="percentage">
    a
  </output>
  <operator>=</operator>
  <function type="window">
    <name>slope</name>
    <input type="spatial" valuescale="scalar">
      b
    </input>
  </function>
</statement>
```

The explicit definition is more verbose and contains more information than the original definition. Apart from explicitly stating that the function used is called `slope` it is also stated that the output data is a spatial with scalar values.

5 TOOL ARCHITECTURE

By converting the model script to an explicit definition some of the modelling tools can use the information which is implicitly present in the model script. While the model script can be regarded as the interface from the model definition to the user, the explicit definition can be regarded as the interface from the model definition to the tools. The explicit definition can be automatically generated from the model script. This has the following important advantages: 1: the modeller doesn't have to maintain different definitions of the same model and 2: the tools which use the explicit definition, can automatically change their behaviour after the developer changes the model script.

In the original architecture without using the implicit information in an explicit form, only the model engine uses the model script to calculate the output data sets from the input data sets (Figure 2). The other tools need to be configured by the developer each time they are used. In the new situation the model script converter uses the model script to produce the explicit model definition. The other tools, including the model engine, read the explicit definition for the information they can use (Figure 3). This means that modelling tools can configure themselves to the model at hand and that changes in the model are automatically reflected by changes in tool configuration. Since the model script is changed often automatic tool configuration can speed up the creation of a model.

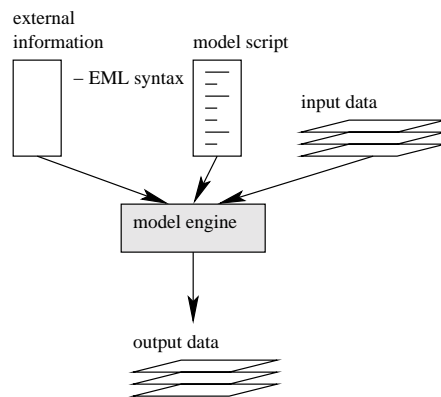


Figure 2: Original architecture.

Technically this means that the model script parser which reads and checks the model script is moved from the model engine to the model script converter and all relevant modelling tools will be provided

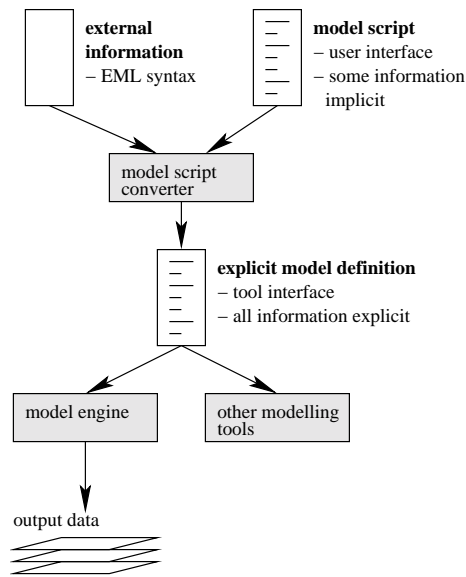


Figure 3: New architecture.

with a parser for the explicit model definition. In our case this definition is an XML file and the modelling tools have an XML parser with which they can query the definition for information. New modelling tools can be written in any programming language provided that they have XML parsing functionality. The XML file itself is platform independent.

6 EXAMPLES OF TOOL CONFIGURATION

In this section, examples are given of how the explicit model definition can be used for tailoring the modelling tools used during the model development cycle.

6.1 Script editor

During model development, a model builder spends a lot of time using the script editor. This tool should assist the modeller and prevent him or her from making syntactical mistakes as much as possible. Additional to features like syntax highlighting, syntax checks, context sensitive help and auto completion the editor can give the modeller script-specific information. As an example of this, the editor can provide the developer with a list of valid input variables according to the properties of the function he or she tries to use. This would prevent the developer for example from writing statements in which the slope of a boolean map is calculated.

```

land = dem > 0;          # Boolean map, height > 0.
gradient = slope(land); # Error!
  
```

This way the modeller is guarded against making the most obvious semantical mistakes.

6.2 Visualisation and EDA

By reading the explicit model definition the visualisation and EDA tools can automatically set up some standard configurations from which the developer can choose. For example, during development the developer often wants to see the input data sets, the output data sets or the data sets linked to a certain function. The following examples demonstrate how this can work in case of a visualisation tool called `visualise` which is started from the command line. Equivalent functionality can be available from a menu of a model explorer environment for example.

```

# All input data from model runoff.
$ visualise --inputs runoff.mod
# All output data.
$ visualise --outputs runoff.mod
# All data linked to slope function.
$ visualise --function slope runoff.mod
  
```

When the model is dynamic, the tools can support animation of output data. Simulations can be compared when the model is stochastic. And the EDA tool can visualise the summary statistics and link elements in a scatter-plot to the individual simulations for example.

6.3 Documentation

When the model is redefined the documentation of the model gets out of sync with it. Apart from the reasoning behind the model structure, all characteristics of it can be automatically described by reading the explicit definition. A documentation tool can be used every time the documentation must be updated. Among other things the following information can be present in the generated document: a description of the required input data sets, whether the model is static or dynamic, a description of the stochastic parameters and a description of the output datasets.

In general, part of the model documentation is redundant: it is a translation in human language of model statements which are written in EML. By generating this documentation automatically errors can be prevented.

7 DISCUSSION

The described architecture has the advantage that the Environmental Modelling Language (EML) and the language used to implement the explicit definition (the XML application in our case) can evolve independently when needed. It is possible that the file format used for the explicit definition evolves and more tools start to support it. In that case the model developer would have a choice of tools to use (given the fact that the tools also support the data file formats used). This differs from the current situation where most modelling tools are specific to the modelling environment.

Another consequence is that the EML is likely to be extended to better support the modelling tools besides the model engine. For example, to support calibration of parameters, value ranges for those parameters could be added to the model definition, to restrict the range of values in which the calibration tool will search for the optimal value.

Additionally, by using the explicit model definition tools can be developed which would otherwise be more difficult to implement. For example, as described in Section 2, model users often need easy to use graphical user interfaces to control the model and a tool has been developed to automatically create such an interface by reading the explicit model definition. Using this default interface as a starting point, the model developer can fine-tune it to allow certain model parameters to be adjusted for example. And when the model definition changes, the interface changes too.

A final point to make about the presented approach is that if the source code of the modelling tools is not available, none of the improvements can be realised. Changing a modelling tool to support the explicit model definition requires at least adjustments to the code which starts and initializes the tool.

8 CONCLUSIONS

Environmental model development is an iterative process involving the different phases of the model development cycle. Typically, different tools are used to create and execute the model, analyse model data, and calibrate and validate the model. In order to improve the model development cycle we have shown that the model script contains information which is useful for configuring some of the tools used. Currently the model definition is only used as input for the model engine which executes the

model. By converting the model script to an explicit model definition in which all implicit information is made explicit it can be used to automatically tailor some of the tools used during model development to the model at hand, improving the model development cycle.

REFERENCES

- AML. ARC Macro Language, Environmental Systems Research Institute (ESRI). <http://www.esri.com>, April 2002. Macro language which is part of the ARC/INFO GIS software.
- Karssenbergh, D. Higher level programming for distributed hydrological model development. *in press: Hydrological Processes*, 2002.
- PCRaster. PCRaster Environmental Software (PES). <http://www.pcraster.nl>, April 2002.
- PEST. S.S. Papadopoulos & Associates, Inc (SSPA). <http://www.sspa.com/pest>, April 2002. SSPA distributes PEST.
- STELLA. High Performance System, Inc. <http://www.hps-inc.com>, April 2002.
- Van der Perk, M. Effect of model structure on the accuracy and uncertainty of results from water quality models. *Hydrological Processes*, 11:227–239, 1997.
- Wesseling, C., D. Karssenbergh, W. Van Deursen, and P. Burrough. Integrating dynamic environmental models in GIS: the development of a dynamic modelling language. *Transactions in GIS*, 1:40–48, 1996.
- XML. Extensible Markup Language, World Wide Web Consortium (W3C). <http://www.w3.org/XML>, April 2002.