ENDURE

European Network for Durable Exploitation of crop protection strategies

Project number: 031499

Network of Excellence
Sixth Framework Programme

Thematic Priority 5
FOOD and Quality and Safety

Deliverable DR4.15

Modelling-tool software package for end-users and developers

Due date of deliverable: M41

Actual submission date: M42

Start date of the project: January 1st, 2007 **Duration**: 48 months

Organisation name of lead contractor: Aarhus University

Revision: V1

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)	
Dissemination Level	
PU Public	**PU**
PP Restricted to other programme participants (including the Commission Services)	
RE Restricted to a group specified by the consortium (including the Commission Services)	
CO Confidential, only for members of the consortium (including the Commission Services)	

Table of contents

Glossary

AU	Aarhus University, Denmark
CNR	Italian National Research Council, Italy
ENDURE	European Network for Durable Exploitation of crop protection strategies
GPL	GNU General Public License [www.gnu.org/copyleft/gpl.html]
GUI	Graphical User Interface
IHAR	Plant Breeding and Acclimatization Institute, Polans
INRA	French National Institute for Agricultural Research, France
PRI	Plant Research International, Netherlands
SSSUP	Scuola Superiore Sant'Anna, Italy
RRES	Rothamsted International, UK
SZIE	Szent István University, Hungary
UniSim	Universal Simulator Software
USDA	United States Department of Agriculture, USA
XML	Extensible Markup Language [www.w3.org/xml]

Summary

1.1. Objectives

To develop a generic, open-source software package useful for research and education in ecological modelling.

1.2. Rationale

Mathematical modelling is often used in ecology to extract and hypothesize general relationships from the complexity of the system under study. But even though modelling is an apt scientific tool, the sad fate of most ecological models is that they were developed, published and soon forgotten. Rarely do we see an example of a model being re-used, and a model being developed by one research group and then used by another is a rare encounter indeed.

To improve this situation we developed a new tool: *Universal Simulator* (*UniSim*) a software package for collaborative ecological modelling. It is composed of a GUI main module which is used to open and execute model specifications read from XML files. The XML files specify the components constituting a model. The functionality of these components are defined in plug-in libraries. This makes UniSim extendible and open for re-use. It is programmed in standard C++ but relies on the Qt library. Universal Simulator is open source (GPL) and publicly available from www.ecolmod.org.

1.3. Degree of validation and operability of findings

UniSim is already used in research for modelling perennial weeds (AU together with PRI and RRES), annual weeds (AU together with SSSUP) and insect pests (AU together with USDA).

UniSim is used regularly as a teaching tool at AU, both at graduate and post-graduate level. It will be used as a teaching tool at international workshops, July 2010 in Hungary and September 2010 in Tanzania.

1.4. Teams involved

AU, CNR, IHAR, INRA, PRI, SSSUP, RRES, SZIE, USDA.

1.5. Geographical areas covered

Currently applied in Europe, Africa and USA. The tool is generic and not bound by geography.

2. State of the art

The study of weed demography is essential to weed science as it helps to understand one of the most troublesome features of weeds, namely their persistence. Mathematical modelling is often used, in weed science as in ecology in general, to extract and hypothesize general relationships from the complexity of the system under study. But even though modelling is an apt tool for weed science, the sad fate of most weed demographic models, recently counted at more than 100 (Holst et al. 2007), is that they were developed, published and soon forgotten. Rarely do we see an example of a model being re-used, and a model being developed by one research group and then used by another is a rare encounter indeed. It is as if all these modellers worked in isolation, developing every little piece of these models again and again, despite the many commonalities between their models. It is symptomatic for the lack of an appropriate, common modelling tool that weed models are still being written in all sorts of programming languages and software. Weed modellers do have access to general modelling tools. They are offered as part of many software packages. But these packages have been written primarily for and by engineers, not plant ecologists. Hence no modelling tools are at hand for the weed modeller, matching her often limited skills in mathematics and programming, and her special problem domain: agronomy, ecology and economy.

What weed modellers need is a standardized set of tools that will enable them to collaborate and share model components, pieces of well-defined software objects that can be interchanged and combined freely: Download the annual life cycle models of species *A* and *B*, combine them with soil model *C* and weather generator *D*, and set it all in the context of cropping system model *E*. We need a way out of modelling as a purely personal, or in the best case, small-team exercise.

A first step in the development of any model is to realize and define for what purpose the model is intended. The same goes for the development of a modelling tool. Which questions will the models developed with this tool address? WeedML (‘Weed Markup Language’) presented in the forthcoming paper *WeedML: a Tool for Collaborative Weed Demographic Modeling* (Holst, 2010) was designed to ease the development of weed demographic models, providing both a conceptual framework to ease collaboration and a computational framework facilitating the sharing and re-use of models, in part or as a whole.

The intended use of WeedML models is research and education. Thus WeedML is intended as a tool for weed *science* rather than weed *technology*. The indiscriminate mixing of weed science with weed technology disrupts progress in either discipline (Cousens 1999), whether modelling is involved or not, and may be one of the reasons why the usefulness of demographic models in weed technology remains to be demonstrated (Freckleton et al. 2008;Moss 2008). Most often weed demographic models are constructed with the explicit purpose of stimulating innovation within weed technology, rather than creating new knowledge in weed science, even though the latter seems to be the true driving motive (Holst et al. 2007). A model may, for example, claim to be a tool for ‘finding optimal solutions’ or ‘designing new control strategies’. But no farmer would dare taking into practice solutions underpinned by computational artifact only.

The development of WeedML was inspired by SBML (‘Systems Biology Markup Language’), a language for representation and exchange of biochemical network models (Hucka et al. 2003). Both WeedML and SBML are dialects of the universal XML standard: ‘Extensible Markup Language (XML) is a simple, very flexible text format ... playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere' (www.w3.org/xml). Initially the authors behind SBML offered only a small selection of software to handle and execute models written in SBML. Their bold initiative could have failed, if systems biology researchers and software engineers had ignored their strive for standardization, but it was in fact so well received that by June 2009, 210 SBML models and 170 software packages supporting SBML were accessible at the community web site (www.sbml.org).

WeedML, as described by Holst (2010), was developed as a tool to model weed demography but looking back at the finished design, it was obviously as well suited also to model other complex systems. This universality is facilitated by the openness that the library system gives: any models or output options, or alternative methods of integration, can be added as new libraries. The open library structure is also a possible vulnerability. WeedML comes with only a limited set of models in its libraries and will only be of general utility, if modellers will enrich the first proof-of-concept libraries with new ones to extend the selection of available models. Reflecting upon the expressiveness of WeedML it was decided to call the accompanying software 'Universal Simulator' and, indeed, libraries already exist for modelling insects as well as weeds (www.ecolmod.org).

Literature cited

Cousens, R. D. 1999. Weed science doesn't have to be a contradiction in terms! 364-373 *in*, Twelfth Australian Weeds Conference. Hobart, Tasmania: Tasmanian Weed Society.

Freckleton, R. P., W. J. Sutherland, A. R. Watkinson, and P. A. Stephens. 2008, Modelling the effects of management on population dynamics: some lessons from annual weeds. Journal of Applied Ecology 45:1050-1058.

Holst, N. 2010. WeedML: a tool for collaborative weed demographic Modeling. Weed Science, *in press*.

Holst, N., I. A. Rasmussen, and L. Bastiaans. 2007, Field weed population dynamics: a review of model approaches and applications. Weed Research 47:1-14.

Hucka, M., A. Finney *et al*. 2003, The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. Bioinformatics 19:524-531.

Moss, S. R. 2008, Weed research: is it delivering what it should? Weed Research 48:389-393.

3. Harmonization of material and methods among the Network

The Universal Simulator was designed as a tool for collaboration. It can be used as a tool to harmonize different model concepts and designs among partners in research and education. It is planned to be used for this purpose in an upcoming EU project.

4. Results

Universal Simulator is available in the form of the following products:

- End-uder installation file for Windows (Linux and Mac OS versions scheduled for late 2010) [www.ecolmod.org].
- Source code (C++, XML) for developers [www.github.com/NielsHolst/UniSim].
- A tutorial *Universal Simulator Explained* [www.ecolmod.org]. Work in progress can be downloaded; introductory chapters finished.
- PhD course *Ecological Modelling* based on Universal Simulator, running 2010 at AU. An e-learning version scheduled for 2011.

5. Road map

The continued development of Universal Simulator has been secured by new EU and Danish project funds. In addition proposals aiming for USA funds are in preparation.

Conclusion

- A generic, open-source software package *Universal Simulator* for research and education in ecological modelling has been made available.

- *Universal Simulator* has raised international interest among ecological modellers in both Europe, Africa and USA.

- *Universal Simulator* has been approved internationally, as testified by both, new projects and projects in preparation, based on Universal Simulator.

Appendix A

WeedML: a Tool for Collaborative Weed Demographic Modeling

Niels Holst*

*Aarhus University, Faculty of Agricultural Sciences, Department of Integrated Pest Management, Flakkebjerg, 4200 Slagelse, Denmark. mail: niels.holst@agrsci.dk.

WeedML is a proposed standard to formulate models of weed demography, or maybe even complex models in general, that are both transparent and straightforward to re-use as building blocks for new models. The paper describes the design and thoughts behind WeedML which relies on XML and object-oriented systems development. Proof-of-concept software is provided as open-source C++ code and executables that can be downloaded freely.

Key words: simulation, object-oriented, XML

The study of weed demography is essential to weed science as it helps to understand one of the most troublesome features of weeds, namely their persistence. Mathematical modeling is often used, in weed science as in ecology in general, to extract and hypothesize general relationships from the complexity of the system under study. But even though modeling is an apt tool for weed science, the sad fate of most weed demographic models, recently counted at more than 100 (Holst et al. 2007), is that they were developed, published and soon forgotten. Rarely do we see an example of a model being re-used, and a model being developed by one research group and then used by another is a rare encounter indeed. It is as if all these modelers worked in isolation, developing every little piece of these models again and again, despite the many commonalities between their models. It is symptomatic for the lack of an appropriate, common modeling tool that weed models are still being written in all sorts of programming languages and software. Weed modelers do have access to general modeling tools. They are offered as part of many software packages. But these packages have been written primarily for and by engineers, not plant ecologists. Hence no modeling tools are at hand for the weed modeler, matching her often limited skills in mathematics and programming, and her special problem domain: agronomy, ecology and economy.

What weed modelers need is a standardized set of tools that will enable them to collaborate and share model components, pieces of well-defined software objects that can be interchanged and combined freely: Download the annual life cycle models of species A and B, combine them with soil model C and weather generator D, and set it all in the context of cropping system model E. We need a way out of modeling as a purely personal, or in the best case, small-team exercise.

A first step in the development of any model is to realize and define for what purpose the model is intended. The same goes for the development of a modeling tool. Which questions will the models developed with this tool address? WeedML ('Weed Markup Language') presented in this paper was designed to ease the development of weed demographic models, providing both a conceptual framework to ease collaboration and a computational framework facilitating the sharing and re-use of models, in part or as a whole.

The intended use of WeedML models is research and education. Thus WeedML is intended as a tool for weed *science* rather than weed *technology*. The indiscriminate mixing of weed science with weed technology disrupts progress in either discipline (Cousens 1999), whether modeling is involved or not, and may be one of the reasons why the usefulness of demographic models in weed technology remains to be demonstrated (Freckleton et al. 2008;Moss 2008). Most often weed demographic models are constructed with the explicit purpose of stimulating innovation within weed technology, rather than creating new knowledge in weed science, even though the latter seems to be the true driving motive (Holst et al. 2007). A model may, for example, claim to be a tool for 'finding optimal solutions' or 'designing new control strategies'. But no farmer would dare taking into practice solutions underpinned by computational artifact only.

The development of WeedML was inspired by SBML ('Systems Biology Markup Language'), a language for representation and exchange of biochemical network models (Hucka et al. 2003). Both WeedML and SBML are dialects of the universal XML standard: 'Extensible Markup Language (XML) is a simple, very flexible text format ... playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere' (www.w3.org/xml). Initially the authors behind SBML offered only a small selection of software to handle and execute models written in SBML. Their bold initiative could have failed, if systems biology researchers and software engineers had ignored their strive for standardization, but it was in fact so well received that by June 2009, 210 SBML models and 170 software packages supporting SBML were accessible at the community web site (www.sbml.org).

This paper is an introduction to WeedML. It describes by example the thoughts behind its design, its syntax and use. It is a hopeful introduction. WeedML could not possibly match the speed at which SBML was suddenly available in a variety of software packages; weed ecology is an economically less potent discipline than systems biology. Weed modelers are internationally a small group but that makes it even more important to draw together the limited resources. This paper aims to show how a particular software solution (WeedML) can help to tame the complexity of weed demography models and, in general, demonstrate how modern software design principles can be applied to ecological modeling.

WeedML Design

XML Documents. WeedML documents adheres to the rules for XML documents (e.g. Kay 2004). Following the first line, that essentially tells which character set is used, an XML document consists of a hierarchy of elements:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<animals>
    <reptiles eggs=”yes”>
        <snakes>
            <cobra/>
            <mamba/>
        </snakes>
    </reptiles>
    <mammals eggs=”no”>
        <elephant size=”big”/>
    </mammals>
</animals>
```

The structure is laid out by start-end tags, e.g. <snakes> ... </ snakes >. In empty elements the tags can be combined, e.g. <cobra/>. The outermost element (‘animals’ above) is called the document node; it contains all other elements in the document. An element holding other elements inside is called the ‘parent’ of those (called ‘children’). In the example, snakes is the parent of cobra and mamba. Elements can be equipped with attribute-value pairs, e.g. attribute ‘size’ with value ‘big’. WeedML uses no other features of XML but these.

WeedML Documents. WeedML defines a vocabulary of valid elements and attributes and flexible rules for how they can be combined to describe a weed demography model. Fig. 1 outlines a sample WeedML document.

A model element can hold additional model elements, nested to any level, plus parameter elements which hold the parameter values needed by the model. The integrator element determines the way the simulation is carried out. Output elements tell how to present simulation results, e.g. on screen. Both integrator and presentation elements hold additional kinds of elements to specify their behavior.

WeedML Elements. A WeedML document must be interpreted by special software. During interpretation this software turns the document into live software objects, representing the elements listed in the document: models, integrator and outputs and any elements inside them. The document specifies the full list of elements which together and by help of the interpreter will carry out the desired simulation.

One should think of software objects as entities that only live in computer memory during the simulation. The objects have different behavior depending on their kind, and they can interact by sending messages to each other. The interpreter orchestrates all this in a consistent and well-defined manner. The class of a model (e.g. ‘annual weed’) determines which kind of software object will be created and thus defines its behavior during execution. The name of a model simple functions as an identifier of the model. For convenience all names are interpreted without regards to case. Furthermore, spaces, hyphens and underscores are simply ignored.

A WeedML document should hold one integrator element, which coordinates the execution of the models defined in the document. Different classes of integrators can direct execution to run once, or maybe several times for sensitivity analysis or stochastic simulation. WeedML output elements simply define which state variables should be presented in output and how.

Element Libraries. The WeedML language was designed to be flexible and extendible. In itself it defines very little functionality. The functionality comes in the form of software libraries for model, integrator and output elements. A library defines a vocabulary of classes. A library of models could, for example, define the classes: annual weed, weather, soil, etc. Each class is identified by a unique name inside the library. A library defines the behavior, intended use and requirements for each of the classes in its vocabulary. The available element libraries thus set the limits for what can be formulated in a WeedML document, and a WeedML document only makes sense in the context of the present libraries.

Ideally, libraries should be self-documenting, so that they can generate a list of their classes and a short description of their functionality and interface. The interface of a model tells (1) which other models it depends on, (2) which state variables it has (which other models can query during execution) and (3) which input variables it has (whereby other models can pass it information during execution). For example, a minimal weed model would likely (1) depend on the presence of a weather model (which it can query daily about temperature and precipitation), (2) possess state variables for leaf area index, biomass and growth stage, and (3) accept input on imposed mortality and growth reduction caused by control treatments.

If the vocabulary of different libraries overlap, i.e. if classes with the same name occur among them, names can be qualified by the library name. For example two model classes both named ‘competition’, but residing in two different libraries ‘Aarhus’ and ‘Wageningen’, can be distinguished in WeedML documents by writing ‘Aarhus::competition’ or ‘Wageningen::competition’.

Parameters. Models will, depending on their class, need to know some parameter values. These are specified by parameter elements inside the model elements. For example, to define parameter ‘a’ with the value 35, you would put this element inside the model:

```
<parameter name="a" value="35"/>
```

The value of a parameter can be given in four different ways:

(1) Atomic, e.g. “35” or “Paris”.

(2) List, e.g. “(1 1 2 3 5 8)” or “((1 10)(2 50)(8 75))”.

(3) Distribution, e.g. “$normal(avg=54.3, sd=12.42)”.

(4) Lookup, e.g. “$lookup(../common/parameter[@name=’duration’]/@value)”.

An atomic value is just a single value. Lists of values are written in parentheses which can be nested. A distribution of values is defined by a statistical distribution and its parameters; distributed values are used for stochastic modeling. A lookup value finds a value defined elsewhere, using the XPath language (www.w3.org). XPath is a powerful language but usually only a few of its facilities are useful in WeedML. The example above looks into the parent element (“..”), selects the child called ‘common’ and inside that selects the parameter element with an attribute called ‘duration’ and for that selects the value of the attribute called ‘value’. Functions are preceded by a ‘$’.

WeedML Re-use. Existing WeedML elements can be re-used simply by declaring that an element is based on another element, which can reside either in the same document or in an XML document elsewhere, on the local computer or on Internet. The element is looked up using XPath:

```
<model name="Galium aparine"
    base="doc(www.ecolmod.org/models/annual-weed-models.xml)/model[@name='galap']">

    <model class=" plant growth stage " name="flowering"
        <model class="reproduction" >
            <parameter name="slope" value="120"/>
        </model>
    </model>

</model>
```

We cannot tell from this of which class 'Galium aparine' is but we can see that it is looked up in an XML document residing in a folder of the www.ecolmod.org site. The value of the 'base' attribute is interpreted as an XPath query, and the result of the query is copied into the 'Galium aparine' element. Hence all the models and parameters found by the query will be filled into the 'Galium aparine' model. Any models or parameters defined locally (in this case, the 'slope' parameter and the models in which it resides) will override what was filled in from the base model. Overriding can mean either over-writing (if models and parameters already existed in the base model) or expanding (if they did not exist in the base model).

This facility, inspired by object-oriented programming, results in rich possibilities for reuse of whole or parts of models (Stroustrup 1997). The other WeedML elements, integrator and output, can be derived from base elements in the same fashion.

Resources. A choice of WeedML libraries, fully functional as software building blocks, and software to read WeedML files and execute them in the context of those libraries can be downloaded from the web site: www.ecolmod.org. The open source software ('Universal Simulator') comes with tutorials built around example models and takes prospective users on a walk through their development in C++ and WeedML code.

WeedML Elements

This section shows by example how a WeedML document can be constructed and how different features of WeedML can be used to assemble models.

Weed Model. The sample WeedML document (Fig. 1) begins with an annual weed model which holds three life stages (seed, juvenile and flowering) in their natural order:

```
<model class="annual weed" name="galap">
    <model class="seed bank">
        <parameter name="initial density" value="1000"/>
        <parameter name="yearly mortality rate" value="0.1"/>
        <parameter name="yearly emergence rate" value="0.2"/>
        <parameter name="emergence calendar" value ="(0 5 30 60 70 50 10 0 0 0 0 0)"/>
    </model>

    <model class="plant growth stage" name="juvenile"
        <model class="day-degrees">
            <parameter name="threshold" value="2"/>
        </model>
        <parameter name="duration" value="540"/>
    </model>

    <model class=" plant growth stage " name="flowering"
```

```
            <model class=”day-degrees”>
                <parameter name=”threshold” value=”4”/>
            </model>
            <parameter name="duration" value="110"/>
            <model class="reproduction" >
                <parameter name="slope" value="56"/>
            </model>
        </model>
    </model>
```

The seed stage model has four parameters, on of which is a list of 12 numbers defining the relative emergence in each month. The models for juvenile and flowering plants hold parameters for stage duration measured in day-degrees, as defined by the ‘day-degrees’ models. The flowering stage in addition holds a model for reproduction.

As already mentioned, the exact functionality of classes is not defined in the WeedML document but in software libraries that implement and document the classes. If WeedML libraries are designed sensibly they will contain classes that are easy to understand and re-use. A minimal documentation could for example read thus:

‘An *annual weed* model consists of a *seed bank* model followed by a sequence of *plant growth stage* models. A *seed bank* model has an input variable to accept new seeds and a state variable holding the current density. A *plant growth stage* model must hold a model with a *time* interface, e.g. of class *day-degrees* or *effective-day-degrees*, which will specify the times units of its duration, otherwise a chronological time scale will be assumed. A *time* model may need access to a *weather* model to inquire about the daily average temperature. A *plant growth stage* holds an input variable to accept an inflow of plant material and state variables to represent the current outflow rate and current density. The input and state variables exists in units both of individuals and biomass. A *plant growth stage* can hold a *seed production* model, which holds a state variable for the current rate of seed production.’

Weather Model. For the service of other models, a weather model must be specified:

```
    <model class="weather file" name=”weather”>
        <parameter name="file name" value="paris 2001.txt"/>
        <parameter name="column date" value="1"/>
        <parameter name="column avg temp" value="2"/>
        <parameter name="column min temp" value ="3"/>
        <parameter name="column max temp" value ="5"/>
        <parameter name="column total irradiation" value ="7"/>
    </model>
```

In this example daily weather is provided from a weather log file, in which climatic variables are available from the designated columns. Alternative weather models could hold a constant environment or contain whole weather generators. Notice that, in any case, the weather model would be equipped with the same minimum set of state variables, representing the current values of the climatic variables. In other words, all these different classes would conform to the same interface, and other models in the WeedML document would be ignorant about which specific class of weather model were in effect.

External Crop Model. Many crop models already exist which could be re-implemented as WeedML models. Here we will not present any particular model but instead assume it already exists and can simply be re-used:

```
<model name="winter wheat"
     base="doc(www.ecolmod.org/models/simple-crop-models.xml)/model[@name='winter wheat']">
</model>
```

Competition Model. A competition model is an example of a model that coordinates the interaction between other models, in this case crop and weed models. Here we will use the recursive density equivalents model of competition, suggested by Holst (2005):

```
<model class="recursive density equivalents competition">
    <parameter name=”ai” value=”((winter-wheat galap 80 0.6) (winter-wheat cheal 35 0.4))”/>
</model>
```

A parameter list defines the hyperbolic coefficients (*a* and *i*) of Cousens (1985) for each crop-weed interaction.

WeedML Integrator. A WeedML document comes to life when it is read by a WeedML interpreter. This software must first read the WeedML code and resolve all internal and external references. Then all elements must be represented internally in software, most simple by creation of one software object for each WeedML element. These objects must be equipped with the needed references to each other (e.g., from competition model to competitors) and they must be set to a well-defined initial state. The state of an object will be held by state variables as needed for the object’s functionality. The initial value of state variables can either have some natural value (often zero) or be specified by WeedML parameters.

Finally, the model can be executed (‘run’) in a loop that step by step updates all state variables of all objects constituting the model. Often a time step of one day is natural and yields a reasonable short execution time. For every time step, the increment or decrement of all state variables (i.e., the rates of change) are, mathematically ideally, calculated before the state variables are actually changed. To make this possible, however, certain mathematical constraints would have to be imposed on the function of WeedML models. A typical problem would be to ensure that several loss rates did not add up to a greater loss than what is present.

Therefore, to give the developers of WeedML greater flexibility, at the loss of mathematical precision, the execution of WeedML models is based upon a sequence of calculations for every time step, in which it is allowed to change the value of state variables before all rates of change have been calculated. To keep model behavior well-defined, a certain order of calculation must then be defined. This sequence of calculation is defined in the WeedML integrator element:

```
< integrator class="single run">
    <parameter name="duration" value="365"/>
    <sequence>
        <model name="weather"/>
        <model name="winter wheat"/>
        <model class="annual weed"/>
    </sequence>
</ integrator >
```

Here, in every time step, the models are updated in the order of (1) models named ‘weather’, (2) models named ‘winter wheat’, and (3) models of class ‘annual weed’. Most often there will be a natural sequence of calculations, e.g. the climatic variables are updated before the models using them. Parent models will update their children in a recursive fashion, e.g. the child and grandchild models inside the annual weed models.

Integrators can have more or less complicated behavior depending on their class, e.g. to perform sensitivity analysis or stochastic simulation, but in this example the model is just run once for 365 days. The classes of integrators available depends on which are offered in the present integrator libraries.

WeedML Output. The results of the simulation can be shown as output of state variables, either as plots on screen or as tables in text files. The example show how to define a plot on screen:

```
<output class="plot">
    <parameter name=”title” value="Crop leaf area index"/>
    <variable axis=”x” label="Date" value="calendar[date]" />
    <variable axis=”y” label="Temp (oC)" value="weather[avg temperature]"/>
    <variable axis=”y” label="Crop LAI" value="winter wheat[lai]"/>
</output>
```

This will produce one plot on the screen with calendar date on the x-axis and two curves: one for temperature and one for crop leaf area index. State variables are specified by a model name followed by the state variable name in brackets. Which state variables a model offers depends on its class. A list of state variables for all model classes should therefore be a part of the documentation for any WeedML library.

Discussion

WeedML is based on a few generic building blocks, yet is expressive enough to formulate complex models of weed demography. By facilitating re-use of models in a modular fashion, WeedML enables modelers to concentrate on the system components of particular interest; models for the remaining parts of the system can simply be borrowed from other modelers with other interests and focus.

WeedML was developed as a tool to model weed demography but looking back at the finished design, it is obviously as well suited also to model other complex systems. This universality is facilitated by the openness that the library system gives: any models or output options, or alternative methods of integration, can be added as new libraries. The open library structure is also a possible vulnerability. WeedML comes with only a limited set of models in its libraries and will only be of general utility, if modelers will enrich the first proof-of-concept libraries with new ones to extend the selection of available models. Reflecting upon the expressiveness of WeedML it was decided to call the accompanying software ‘Universal Simulator’ and, indeed, libraries already exist for modeling insects as well as weeds (www.ecolmod.org).

So, is WeedML the right tool for any model of weed demography? Certainly most benefits will be reaped from WeedML, if the model is complex and maybe developed by a larger team. For small models, or models that can be formulated rigorously as a system of differential equations or Leslie matrices, other software tools are better suited. Such classical mathematical models can easily be constructed with these tools, which moreover offer plenty of options to analyze the models analytically through mathematical inference. But if one wants to encourage the re-use of even the simplest model, a WeedML version would increase the chance of its more widespread use.

When ecological models become complex, and models of weed demography easily do, the construction of their software representation becomes a genuine software development project. In most cases, modelers seem ignorant, knowingly or not, that they are crossing the border to another science, namely that of software engineering. Maybe this dawns on them, e.g. when they have finished their one crop-one weed model and want to extend this to a crop rotation-many weed species model: the complexity of the software itself becomes unmanageable, even though their idea of the model stands clear in their mind and is easily explained. WeedML will certainly be a help in such cases because in WeedML, the modeler can more easily focus on the programming of individual sub-models, one at a time.

It is interesting to compare the design of WeedML with that of two other modeling tools for biology: HERMES (Larkin and Carruthers 1989;Larkin et al. 2000) and SBML (Hucka et al. 2003). HERMES is a fully-integrated, visual modeling tool. Models and parameter values are all defined with the visual design tool, and additional model building blocks can be programmed in the Smalltalk language. HERMES does not seem in widespread use. SBML is a pure XML standard. Models are specified as differential equations and written out in XML documents down to the detail of equations and parameter values. SBML is highly successful with a plethora of software available to construct and analyze SBML models.

Both HERMES and SBML models run as interpreted code. WeedML takes another approach: it is a language that specifies how compiled building blocks (models, integrator and outputs) are put together to be executed as one coherent model. The Universal Simulator software (www.ecolmod.org) that accompanies this paper was programmed in C++ using the plug-in facility of the Qt framework (www.qtsoftware.com) to implement the lose coupling of building blocks supplied as dynamic-link libraries. Although WeedML tools could be implemented in any language, the most straightforward way of creating new libraries for WeedML will certainly be to use the same toolset, neatly integrated in the free tool Qt Creator (www.qtsoftware.com).

The WeedML development method is first to create the needed building blocks (i.e. WeedML models), using e.g. Qt Creator and the framework of WeedML base classes found at www.ecolmod.org. These models must be small, well-formed pieces of functionality with clear interfaces to other models. Thereby WeedML encourages good software design (Martin 2006). Once the models have been implemented in a library they are assembled to form the full model. This two-stage process helps managing the complexity of models with many interacting parts: one can add more weed species or more crops to the rotation without losing oversight of model functionality. Or a simply model of e.g. weed emergence can be replaced with a more complicated one without the fear of unwanted side effects in other models.

The spirit of WeedML is open source software and the Universal Simulator (www.ecolmod.org) indeed is open source. When WeedML models developed by one modeler is re-used by another, it will be necessary to study the source code of that model, if one wishes to understand how it works in detail. Such detailed knowledge will be necessary if the borrowed model is to collaborate intimately with some new model under construction. But this requirement is not unique to WeedML, it is a necessity for any complex simulation model that claims to be a valid scientific contribution. If the source code is not available for scrutiny, scientists from outside will have to trust that the modeling team truthfully implemented the model as described in the scientific paper, and that it did not make any errors when coding the model into software.

Laying model code out in the open invites both re-use, through standards such as WeedML, and communal inspection and debugging of the code. In this, open source models are like scientific papers, which also are put forward boldly to be cited, used and possibly corrected or falsified. An important difference between models and papers is that the former are soft: they can be reformulated and extended *ad infinitum*. It is the experience of the open-source community that this process leads to increased usability of the software rather than its detriment (e.g. Fogel 2003).

WeedML is an attempt to create a standard that will ease collaboration between weed modelers. The success of this approach will depend on the interest of the weed science community, in terms of using WeedML in its current form and in developing it further in terms of software tools and additional WeedML libraries. Should WeedML end up not as a success in itself but as source of inspiration for a better design of weed demographic models, or ecological models in general, that would still be a significant achievement.

Acknowledgments

Thanks to the WeedML team for inspiring discussions. This work was supported by the EU Network of Excellence ENDURE and the International Centre for Research in Organic Food Systems (ICROFS).

Literature Cited

Cousens, R. D. 1985, A simple model relating yield loss to weed density. Annals of Applied Biology 107:239-252.

Cousens, R. D. 1999. Weed science doesn't have to be a contradiction in terms! 364-373 in, Twelfth Australian Weeds Conference. Hobart, Tasmania: Tasmanian Weed Society.

Fogel, K. , 2003. Open Source Development with CVS, 3rd Ed. Paraglyph Press.

Freckleton, R. P., W. J. Sutherland, A. R. Watkinson, and P. A. Stephens. 2008, Modelling the effects of management on population dynamics: some lessons from annual weeds. Journal of Applied Ecology 45:1050-1058.

Holst, N. 2005, Recursive density equivalents: an improved method for forecasting yield loss caused by mixed weed populations. Journal of Agricultural Science 143:293-298.

Holst, N., I. A. Rasmussen, and L. Bastiaans. 2007, Field weed population dynamics: a review of model approaches and applications. Weed Research 47:1-14.

Hucka, M., A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, A. P. Arkin, B. J. Bornstein, D. Bray, A. Cornish-Bowden, A. A. Cuellar, S. Dronov, E. D. Gilles, M. Ginkel, V. Gor, I. I. Goryanin, W. J. Hedley, T. C. Hodgman, J. H. Hofmeyr, P. J. Hunter, N. S. Juty, J. L. Kasberger, A. Kremling, U. Kummer, N. Le Novere, L. M. Loew, D. Lucio, P. Mendes, E. Minch, E. D. Mjolsness, Y. Nakayama, M. R. Nelson, P. F. Nielsen, T. Sakurada, J. C. Schaff, B. E. Shapiro, T. S. Shimizu, H. D. Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, and J. Wang. 2003, The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. Bioinformatics 19:524-531.

Kay, M., 2004. XPath 2.0 Programmer's Reference. Indianapolis, Indiana: Wiley Publishing, Inc.

Larkin, T. S. and R. I. Carruthers. 1989. Development of a hierarchical simulation environment for research biologists. Pages 49-54 in A. Gausch, ed. Object Oriented Simulation. Society for Computer Simulation.

Larkin, T. S., R. I. Carruthers, and B. C. Legaspi. 2000, Two-dimensional distributed delays for simulating two competing biological processes. Transactions of the Society for Computer Simulation International 17:25-33.

Martin, R. C., 2006. Agile Software Development: Principles, Patterns, and Practices. Prentice Hall.

Moss, S. R. 2008, Weed research: is it delivering what it should? Weed Research 48:389-393.

Stroustrup, B., 1997. The C++ Programming Language, 3rd Ed. Addison-Wesley.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<weedml version="1.0">
    <model class="annual weed" name="galap"> ... </model>
    <model class="annual weed" name="cheal"> ... </model>
    <model class="weather" > ... </model>
    <model name="winter wheat" base =”...”> ... </model>
    <model class="recursive density equivalents competition"> ... </model>
    < integrator class="single run">  ... </ integrator >
    <output class="screen"> ... </output>
</weedml>
```

Fig. 1. In a WeedML document, the document node can hold three kinds of elements in any order: model, integrator and output. They all have three optional attributes: class, name and base. Element contents (abbreviated '...') are exemplified in text.