Modelling Framework (SeamFrame) requirements


Partners involved: IDSIA, Alterra, CRA, INRA, LUND, PRI, Simulistics, UBONN, UVM
SEAMLESS integrated project aims at developing an integrated framework that allows ex-ante assessment of agricultural and environmental policies and technological innovations. The framework will have multi-scale capabilities ranging from field and farm to the EU25 and globe; it will be generic, modular and open and using state-of-the-art software. The project is carried out by a consortium of 30 partners, led by Wageningen University (NL).

Email: seamless.office@wur.nl
Internet: www.seamless-ip.org

Authors of this report and contact details

Name: Andrea Rizzoli  
Partner acronym: IDSIA  
Address: IDSIA, Galleria 2, 6928 Manno, Switzerland  
E-mail: andrea@idsia.ch

Name: Mats Svensson  
Partner acronym: LU  
Address: Lund University, PO Box 170, S-22100, Sweden  
E-mail: mats.svensson@lucsus.lu.se

Name: Ed Rowe, former scientist at WU  
Partner acronym: WU  
Present address: CEH Bangor, University of Wales, Orton Building, Deiniol Road, Bangor LL57 2UP, UK  
E-mail: ecro@ceh.ac.uk

Name: Marcello Donatelli,  
Partner acronym: CRA-ISCI  
Address: ISCI, Via di Corticella 133, 40128 Bologna, Italy  
E-mail: m.donatelli@isc.i.it

Name: Robert Muetzelfeldt  
Partner acronym: Simulistics  
Address: Simulistics, 11 Tantallon Place, Edinburgh, EH9 1NZ, Scotland, UK  
E-mail: r.muetzelfeldt@ed.ac.uk

Name: Tamme van der Wal  
Partner acronym: Alterra  
Address: Alterra, PO Box 47, 6700AA Wageningen, the Netherlands  
E-mail: tamme.vanderwal@wur.nl

Name: Frits van Evert  
Partner acronym: PRI  
Address: PRI – Wageningen University, P.O. Box 16, 6700 AA Wageningen, the Netherlands  
E-mail: frits.vanevert@wur.nl

Name: Ferdinando Villa  
Partner acronym: UVM  
Address: University of Vermont, 590 Main Street, Burlington, 05405, Vermont, USA  
E-mail: ferdinando.villa@uvm.edu
Disclaimer 1:

“This publication has been funded under the SEAMLESS integrated project, EU 6th Framework Programme for Research, Technological Development and Demonstration, Priority 1.1.6.3. Global Change and Ecosystems (European Commission, DG Research, contract no. 010036-2). Its content does not represent the official position of the European Commission and is entirely under the responsibility of the authors.”

"The information in this document is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability."

Disclaimer 2:

Within the SEAMLESS project many reports are published. Some of these reports are intended for public use, others are confidential and intended for use within the SEAMLESS consortium only. As a consequence references in the public reports may refer to internal project deliverables that cannot be made public outside the consortium.

When citing this SEAMLESS report, please do so as:

# Table of contents

Table of contents........................................................................................................................................... 5  
General information........................................................................................................................................ 7  
Executive summary......................................................................................................................................... 7  
1 Introduction.................................................................................................................................................... 9  
  1.1 Inspiration.................................................................................................................................................. 9  
  1.2 Key ideas.................................................................................................................................................. 9  
2 Environmental modelling frameworks requirements ....................................................................................... 11  
  2.1 Problems of current modelling tools ...................................................................................................... 11  
  2.2 Tackling the problems.............................................................................................................................. 12  
  2.3 The benefits of the specifications............................................................................................................ 13  
3 SEAMLESS technical framework requirements.......................................................................................... 15  
  3.1 Applications.............................................................................................................................................. 15  
  3.2 SEAMLESS for the policy maker............................................................................................................ 16  
  3.3 SEAMLESS for the farmer....................................................................................................................... 16  
  3.4 SEAMLESS for the model and application developer........................................................................... 17  
  3.5 Other potential use scenarios for SEAMLESS for application and model development................. 20  
  3.6 Some more issues ................................................................................................................................... 21  
4 A comparison of existing modelling frameworks ......................................................................................... 23  
  4.1 TIME – The Invisible Modelling Environment ..................................................................................... 24  
     4.1.1 Requirements matching................................................................................................................. 24  
     4.1.2 Contact details.................................................................................................................................. 26  
  4.2 IMA: the Integrating Modelling Architecture ......................................................................................... 27  
     4.2.1 Requirements matching................................................................................................................. 27  
     4.2.2 Contact details.................................................................................................................................. 29  
  4.3 Modcom.................................................................................................................................................... 31  
     4.3.1 Requirement matching.................................................................................................................... 31  
     4.3.2 Contact details.................................................................................................................................. 33  
  4.4 OpenMI .................................................................................................................................................... 34  
     4.4.1 Requirements matching.................................................................................................................... 34  
     4.4.2 Contact details.................................................................................................................................. 35  
Appendix 1. SEAMLESS requirements and suggested solutions...................................................................... i
General information

Task(s) and Activity code(s): T5.2, Activity 5.2.2
Input from (Task and Activity codes): T1.2, Activity 1.2.1
Output to (Task and Activity codes): T5.2, Activity 5.2.3
Related milestones: M 5.2.2

Executive summary

In modern software engineering, *software frameworks* are fundamental to software development. A software framework provides a set of reusable libraries and classes to build applications. Examples are the Smalltalk model view controller (Deutsch, 1989), or the MacApp, the "Macintosh Application Framework" (http://developer.apple.com/tools/macapp/). More recently, the Java software framework (J2SE and J2EE) and the Microsoft .Net framework are providing easy and widespread access to software frameworks to an unprecedented number of developers.

A *modelling framework* is analogous to a software framework, with the specialisation in providing reusable components for building mathematical models. There are many modelling frameworks on the market, examples are MATLAB, Modelica, and so on (the list cannot be exhaustive, see http://www(idsia.ch/~andrea/simtools.html for a broader view).

An *integrated modelling framework* is an extension of a modelling framework, which supports multiple modelling domains and paradigms. The number of integrated modelling frameworks is considerably more limited, especially if we restrain to the intersection of social, economic and environmental modelling. We list some of the most notable initiatives in the various fields.

**Economic modelling frameworks.** GAMS (general algebraic modelling system, http://www.gams.com) and GTAP (global trade analysis program, http://www.gtap.agecon.purdue.edu ) are some of the most used modelling systems in the agro-economic domain. They can also account for social variables, such as unemployment.

**Environmental modelling frameworks.** If we limit to the agricultural domain, the list is quite limited. There is no ‘real’ framework according to the definition, but APSIM, STICS and CropSyst provide some of the functionalities. When we consider the water management sector, we find many examples, such as TIME (the invisible modelling environment), IMT, OpenMI, and OMS.

Other *modelling software environments* of notable interest are SME, MMS, ICMS, Tarsier, Modcom, Simile, but they are integrated modelling environments, not frameworks. This means that they can be used to perform assessments, analyses, decision support, but they do not provide programming structures such as classes, components, objects, design patterns to be used to create end-user applications.

In this document we will express the requirements imposed by the SEAMLESS project vision on the SEAMFRAME modelling framework and we will critically assess these requirements against a suitable set of existing modelling frameworks. As a result, we shall be able to understand what to pick up and what to drop from these previous experiences in order to improve the trade-off between re-use and innovation, and maximise our users’ satisfaction.
1 Introduction

1.1 Inspiration

The SEAMLESS project has the overall ambition of providing a methodological framework (SEAMLESS-IF) to support the ex-ante analysis of European agricultural management policies at all scales, from market level down to biophysical systems level. The SEAMLESS-IF will enable:

1. Analysis at the full range of scales, whilst focusing on the most important issues emerging at each scale;
2. Analysis of the environmental, economic and social contributions of a multifunctional agriculture towards sustainable rural development and rural viability;
3. Analysis of a broad range of issues and agents of change, such as climate change, environmental policies, rural development options, effects of an enlarging EU, international competition and effects on developing countries.

And this will be made possible thanks to an integrated and operational framework, named SEAMFRAME, with the following specific objectives:

1. To develop and test a multi-perspective set of economic, social and environmental indicators of the sustainability and multifunctionality of systems, policies and innovations in agriculture and agroforestry, and to establish, as far as possible, threshold values for these indicators and/or to enable trade-off analysis.
2. To provide quantitative and qualitative tools and databases for integrated evaluation of agricultural systems at multiple scales and for varying time horizons.
3. To develop a software architecture that allows reusability of model and database components and knowledge, also ensuring transparency of models and procedures developed.

1.2 Key ideas

SEAMFRAME will be based on the following key ideas:

First, the language used to represent models within SEAMFRAME will be declarative and not imperative. This means that a model will be represented as a set of facts and relations that are true about the model, rather than as an implementation in a specific compilable or compiled language. Declarative representation makes models far easier to re-use and combine, and much more transparent than implementations within a traditional, code-based, model structure. Thanks to the declarative modelling approach, the model can be saved in a standard, XML-based model-representation language. A declaratively-represented model can then be used to produce (for example): a description of the model (e.g. HTML); an executable version of the model; a transformation of the model (e.g. to simplify it, thus addressing the scaling problem).

Second, SEAMFRAME will use ontologies (structured specification) of data, models and tools, facilitating their retrieval from digital repositories and integration into workflows (a workflow is a particular arrangement of data, models and tools). Modellers will thus gain transparent access to each other’s work (Note: SEAMFRAME will provide facilities to use the many models that are currently in use, thus preserving past investments and extending an invitation to researchers who are not part of SEAMLESS per se).
Third, SEAMFRAME will provide a mechanism to store comments, conversations and citations about models, data and workflows. An intelligent search agent will enable any user to retrieve and combine information from 1) model specifications, 2) model, data, or workflow metadata and their place in an ontology, 3) comments, conversations and citations. Thus, a researcher faced with the task of modelling a certain system can find not only all suitable models, but also the assumptions underlying those models, experiences of other researchers in using those models, and the outcomes of other studies in which those models were used. A decision maker faced with an analysis in which modelling was used, can use SEAMFRAME to find other studies in which the same models were used, other models that could have been used, and conversations about the history and suitability of the models that were used.
2 Environmental modelling frameworks requirements

Do we need another Modelling Framework? After all, we have a good number of Modelling Frameworks, purposely designed to solve the modelling problems one can encounter in many fields, from economics, to engineering, through natural sciences. Yet, we want another modelling framework, because it looks like we are not satisfied with the current ones.

Software frameworks provide programming constructs and artifacts (e.g. classes and components) that can be re-used to build software applications. Among others, .NET and J2EE are two examples of software frameworks, since they provide a set of libraries and classes with which the programmer can write new applications. Moreover, they provide tools, such as compilers and some of these tools have a graphical interface, such as an Integrated Development Environment.

In the same way, a modelling framework is a set of software components which can be used to assemble a software application for designing and experimenting with models. The libraries and classes are targeted to develop models, which can be packaged into applications. Tools, such as simulators or optimisers, operated on models, and a visual modelling environment can be useful to interactively develop a model or to set up a modelling scenario.

2.1 Problems of current modelling tools

Before listing the requirements of what we expect to be a good modelling framework, let’s briefly summarise the problems we have encountered with the current ones:

1. models are applications. There is nothing inherently bad in this, but the problem is that bundling data, algorithms and the graphical user interface of a model in an application makes the model very hard to re-use out of its original context. Most models are therefore monoliths.

2. Models are nearly impossible to maintain, they can hardly evolve. Once a model, design and implemented as a monolithic software entity has been deployed, its evolution is totally in the hands of the original developers. While this is a good thing for intellectual property rights and in a commercial environment, this is absolutely a bad thing for science and the way it is supposed to progress. Independent revisions and third-party contributions are nearly impossible.

3. Models can not be independently tested. This issue is related to #2, since models often do not come with associated data sets for testing. Moreover, the adoption of object-oriented programming, while it is a good thing for model reusability and portability, it makes things more complex for testing, because of a number of problems such as observability in virtual method calls and state dependent behaviour of objects (Pezzè and Young, 2004).

4. Models are difficult to reuse. This problem is also related to issue #1, since monolithic models display a strong level of internal cohesion, and, if a modeller is interested in reusing a particular function within a bigger model, she can find it very hard to isolate and extract it, given the strong dependencies existing in the source code.

5. Inefficient use of resources. This issue is closely related to the preceding one; given that models are difficult to reuse, modellers and programmers are re-implementing the same things over and over, both in terms of models and tools used to operate on the models.

6. Models are not transparent
7. Models are platform specific

2.2 Tackling the problems

How can we solve the six (but there may be many more) issues we listed above? Many groups are working hard at it, but some of them are hard nuts to crack and it is unavoidable that, for the present time, there will be an overlap of efforts. This can be seen as a waste of resources, but it can also be seen as an evolutionary design effort, where the best ideas and practices (or the most popular) will emerge to provide a common and shared basis for environmental modelling. Moreover, the wide variety of models we encounter in environmental modelling, let alone other modelling domains, will provide food for thought for many researchers in the coming years.

As in any software problem there’s no ‘silver bullet’ (Brooks, 1987). Yet, we have identified a set of approaches and methodologies which have been successfully adopted and used so far in diverse modelling framework efforts:

1. Knowledge management. Knowledge is a vague word, it means everything and nothing. In this context ‘knowledge management’ means to organise knowledge on models and data by means of structuring information. Semantic networks, one of the first attempt to represent knowledge, are an example (see Durkin, 1994 for a general book on Expert Systems and Knowledge Representation). Moreover, the Semantic Web initiative (http://www.w3.org/2001/sw/) is proposing shared web ontologies, as an evolution of semantic networks that can be used to univocally denote modelling concepts.

2. Object-oriented software development. O-O programming is nothing new, but it has proven to be a successful key to the design and implementation of modelling frameworks. Models and data can be seen as objects and therefore they can exploit properties such as polymorphism, data abstraction and inheritance.

3. Design patterns. The adoption of design patterns in programming (Gamma et al., 1995) has been proven to be a major factor for reuse, if not of code, at least of ideas and solutions. A library of design patterns for environmental modelling would be an interesting contribution. The modeller should create new models looking up at simple specifications in the form of patterns and code snippets.

4. Component-oriented software development (see Szyperski et al, 2002). Objects (models and data) should be packaged in components, exposing for re-use only their most important functions. Libraries of components can then be re-used and efficiently integrated across modelling frameworks. Yet, a certain degree of dependency of the model component from the framework can actually hinder reuse.

5. Support for testing. Models should be testable and they should be distributed together with their pre and post conditions and sets of data for testing. Techniques such as unit testing (http://c2.com/cgi/wiki?UnitTest) have been successfully adopted in model development.

6. Support for pre-conditions and post-conditions. It becomes important to adopt a ‘design-by-contract’ approach in modelling, borrowing this concept from software engineering. (Meyer, 1992). Each model will impose pre-conditions on the values of its inputs (e.g. admissible ranges) and post-conditions on the outputs, which can also be expressed as complex logical statements. The use of pre and post-conditions will make possible the use of the testing technique of ‘unit tests’ (Jorgensen and Erickson, 1994, Beck, 2003), enhancing the overall reliability of the system.
7. Support for documentation. Model equations should contain symbols which have their complete meaning in the ontology. The equations could be written in a declarative format (e.g. MathML and/or LaTeX) and then, each symbol could be associated with its meaning.

2.3 The benefits of the specifications

Why should ever a modeller take the trouble of following the complex specifications imposed by a modelling framework? The answer is easy: because he will get much more in return, by making a small investment in learning how to use the framework. Surely, the investment must be moderate and the return must be substantial, otherwise it will be better to continue as usual.

Here we list some of the expected returns, which become the requirements of a modelling framework. The framework shall provide:

- discrete units of software which are re-usable even outside the framework, both for model components and for tools components. Not an easy one to achieve since, as already pointed out, a certain degree of dependency, at least on data structures, is unavoidable;
- declarative modelling: among the multiple benefits, it allows for portability, reuse and openness of the knowledge embedded in models;
- software tools to facilitate model development, structuring and organising modelling ‘knowledge’;
- seamless and transparent access to data, which are made independent of the database layer.
- a number of tools (simulation, calibration, etc.) that the modeller will be free to use (including a visual modelling environment);
- a model repository to store your model and to share it with others.
3 SEAMLESS technical framework requirements

Collection of requirements has started during a meeting held in Wageningen on the 23/24 June 2003. The first set of requirements drove the description of the Technical Framework Working Package within the IP preproposal. Requirements are continuously being collected, being an integral part of the development process based on agile techniques (Beck, 2000).

3.1 Applications

Different applications define the main types of usage for the framework. Three types of user have been defined:

- Prime users: take or prepare decisions at a political level – primarily DGs (Agriculture and Environment) of the European Commission.
- Other end-users: national agencies, representative groups, OECD, etc. They may take or prepare decisions at national or regional level, or represent stakeholder groups.
- Model and application developers/modellers: use the SEAMFRAME components to build models and targeted applications.

These groups have been further detailed according to their roles:

- Coders: implement models, applications and tools.
- Linkers: link existing models and applications.
- Runners: execute existing models, but they create and define scenarios.
- Players: play simulations and experiments comparing scenarios and making analyses.
- Viewers: view the players’ results, have a low level of interaction with the framework.
- Providers: provide inputs and data to all other user roles.

The following table displays a match between user types (rows) and user roles (columns)

<table>
<thead>
<tr>
<th></th>
<th>Coders</th>
<th>Linkers</th>
<th>Runners</th>
<th>Players</th>
<th>Viewers</th>
<th>Providers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prime Users</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Other End-users1</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Technical Users</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1 End-users include all users who are neither modellers (technical users) nor policy makers (Prime Users) and they include national policy development agencies, farmers’ representative groups, etc.
Use scenarios and requirements will be now elaborated for the user types.

3.2 SEAMLESS for the policy maker within DGs – Prime Users

This version of SEAMLESS is a key requirement of the IP call and it is addressed to the Prime Users; supporting policy making through use of social, economic and policy modelling components. Previously developed tools integrating biophysical and economic processes, agent based modelling, participatory decision making, multi criteria decision analysis, etc., will be used at different scales to design, synthesise, and analyse (evaluate) policy impacts at different scales (farm/regional/national/European) and on different sectors (industrial/social/environmental).

Use case 1 (Policymakers): A policymaker (not a modeller) requires a set of tools for evaluating impacts of the newly agreed restructure of the EU CAP from a multicriteria perspective. Several new, not formerly considered aspects will need to be included; rural development, demand for locally produced products, etc. The balance between local, regional and national level effects of policy decisions should be considered. The tools must combine state-of-the-art knowledge from different disciplines (e.g. a crop model with an economic model and a policy model) and yet be transparent and easy to use. The tools must be easy to modify and re-use even if their designers are not present.

This leads to the following requirements:

| A focus on policy-relevant meta-data: | One of the novelties with the SEAMLESS approach is the combining of different types of models for policy evaluation purposes. This creates the need for new meta-data for all types of models included. |
| Application development environment is open-source | Code can be submitted to reviews from external groups. Add-ons and new features can be contributed by groups adopting the SEAMLESS software. Research investments are preserved, since no knowledge is lost: everything is public, accessible and, if needed, can be re-implemented, migrated to other approaches/platforms (see Eric S Raymond article “The Cathedral and the Bazaar” http://www.firstmonday.dk/issues/issue3_3/raymond/) |

3.3 SEAMLESS for Other End-users

In this version of SEAMLESS the user will have more space to build their simulations with (e.g. regional-) specific scenarios, to explore and assess all point of view of various stakeholders.

Use case 2 ( Farmers’ association): A farmers’ representative in a facilitated workshop wants to explore the idea of converting to organic (biological) status. How will this affect the profitability of the farms? What alternative cropping options should be considered? Need to simulate farms via FSSIM. Develop a program and implement models to simulate farm activities in order to provide support for planning, accounting for economic, technical, environmental, and labour perspectives. The farm is seen as a spatially explicit unit, in which different production (in the broad sense) activities may or may not take place: crops, orchards, livestock, forestry. Some of the models will be dynamic, others will be static. There will be an economic budgeting model. Management will account for machinery availability and characteristics, which extends the set of rules to fire events. Need to test FSSIM performance against experimental farm data. Need to evaluate scenarios (settings of resources, factors, and actions with specific or
stochastic weather and on specific soils). Need to validate scenarios before run. Need to save each scenario simulation outputs.

This use-case leads to the following requirements:

<table>
<thead>
<tr>
<th>Easy to use</th>
<th>Specific applications with customised user interfaces will need to be developed for scenario exploration with particular groups of farmers.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliable</td>
<td>Farmers and others directly related to land use issues should be able to pick recommended or “empirically proven” model combinations for sustainability estimations</td>
</tr>
</tbody>
</table>

3.4 SEAMLESS for the model and application developer – Technical Users

This is the version of SEAMLESS which is most sought-after by European scientists and which current SEAMLESS consortium members wish for; an environment for the coherent development of models and their simulation.

The meeting, being software oriented, further specified this application into

1. Development of conceptual models, a tool for modelling
2. Development of “components”. Components can be solvers, numerical integration algorithms, simulation engines or statistical analysis routines, as well as models.
3. Development of tools from these components, that are integrated across scales (temporal and spatial), and disciplines (social sciences, economics, natural sciences).

Use case 3 (Model developers): A research centre wants to use SEAMLESS as a development platform to produce models which are easy to integrate with the work of different research groups. The research group also wants to preserve the investment made in the model development, since language obsolescence often leads to periodical re-implementation of the models.

The requirements emerging from this use case are for an environment which:

- makes it easy to integrate models
- makes it easy to assess and achieve interoperability
- provides methodology for preserving coding investment
- is self-documenting
- facilitates knowledge-based support to modelling
- facilitates a model repository
- facilitates web-based distribution
- facilitates version control
- facilitates use of multiple modelling paradigms
- facilitates spatial and temporal modelling at different scales
- facilitates quality control, through standardised representation, and through easy use of tools for model comparison and testing

All these requirements inevitably lead to a solution, which becomes a requirement itself:
- **A declarative approach to modeling:**

This requires separating a model from the computational aspects which make possible the transformation of the model input into an output. The model contains the “rules” of the transformation, not the algorithm performing the transformation. For instance, in the case of a model based on sets of Ordinary Differential Equations the model consists of the equations, not of the numerical integration routine, the database access routines or the graphical visualisation or statistical processing of the outputs.

From Simile’s website:

> In declarative modelling, we represent a model not as a series of assignment and control statements, but as a set of facts that are true about the model. The order in which we present the facts is (unlike a procedural program) irrelevant. The full set of facts defining a model actually constitute a specification for the model: given these facts, and knowledge about what the symbols mean, someone else can construct a working version of the models.

This implies:

- **The model must be specified using a declarative language.** The language provides a formalism to express the modelling facts. The language provides constructs to define the model interface and the model constructs.

- **The language must be semantically rich.** This requirement implies that the elements of the modelling language must have a meaning that can be understood by a computer. This implies that the language elements must be defined. The repository for these definitions, which are the knowledge about the model, are contained in what is called an ontology, i.e. a schema (or DTD) for the necessary meta-data. Thus a key task for SEAMLESS is to deliver this ontology. Recent research on the Semantic Web (http://www.semanticweb.org) shows how XML and Schemas can be used to represent ontologies. This will enable SEAMLESS to handle an (evolving) ontology, and shall aim at delivering a Schema (and a DTD for creation of XML bindings) for the validation of the necessary meta-data.

In XML (extensible markup language), as with HTML, information is stored together with tags which define what the information is. The difference is that HTML tags are pre-defined, whereas XML tags can be defined in a case-specific ontology. See for example the following piece of XML code; the language elements are tags such as `<INFLOW>`, `<OUTFLOW>` etc.:

```xml
<STOCK NAME = "BUDWORM" LOWER=0>
  <INIT>5</INIT>
  <INFLOW NAME="BIRTHS">
    R*BUDWORM(1-BUDWORM/K)
  </INFLOW>
  <OUTFLOW NAME="DEATHS">
    (BUDWORM**2)/(1+BUDWORM**2)
  </OUTFLOW>
</STOCK>
```


The requirement that modelling must be declarative, and achieved by a semantically rich modelling language, implicitly satisfies a number of other requirements:
| **Models are self-documented** | This is to a certain degree inherent to the DM methodology, since the way the model is written makes clear what is an input, what an output, what a state and what a parameter. DM also makes it easy to develop multiple tools for model analysis and description, for different users / purposes.  

The model of the stock component named “budworm” is clearly self-documenting (even if the declaration of the initial value equal to 5 in the model specification is objectionable).  

Another example of self-documentation (beyond other things) is provided by ECOBAS ([http://www.wiz.uni-kassel.de/ecobas.html](http://www.wiz.uni-kassel.de/ecobas.html)) where models expressed by a declarative approach can be searched, downloaded, compiled, and run. |
| **Knowledge based support to modelling** | This requirement was not expressed in the previous use case, but it was set as a priority by the group. Knowledge-based support comes “for free” once we provide semantic meta-data on model teleology. Teleology is the model purpose and it can be formally expressed via model meta-data (facts associated with the model) and inferred from the input to output transformation. |
| **A model repository** | Models can be stored in a repository and accessed for later usage, queried, compared, evaluated. |
| **The model repository must be under version control** | It should be possible to track changes to the model equations made by different authors and to view a ‘history’ of the development of a model. |
| **Model “quality” must be verifiable where possible** | Here, the word “quality” should be defined according to the actual use. Are we talking about programming structure (e.g. a “good” model is one written clearly)? Are we talking about validated outputs? About how easy it is to use? It must be remarked that many economic and policy models are also hard to validate and / or verify.  

For models of natural systems, this requirement can be satisfied with benchmarking tools that should be provided. Such tools will provide standard statistical tests to assess model quality against data.  

Policy models are harder to benchmark, but some efforts have been done in the field of Participatory Decision Making, using agent based models, trying to reach a consensus based on empirical observations of the agent behaviour.  

*Note: In any case, this requirement is a sensible one and it needs to be discussed together with economists and social scientists.* |
| **Web-based distribution of models** | For an example, see the ECOBAS web site ([http://dino.wiz.uni-kassel.de/ecobas.html](http://dino.wiz.uni-kassel.de/ecobas.html)). Models can be sought after on a website according to their metadata and finally they can be downloaded, either in binary or source code format. |
| **Web based deployment of models** | Models and tools can be packaged together to provide a ‘service’, that is a transformation of data over the internet. |
Multiple modelling paradigms (static/dynamic/stochastic/deterministic/explanatory/qualitative/conceptual). A Lattice of model classes should be listed. (Note: this is also meta-data, which can be covered within the Schema)

Spatial and temporal modelling at different scales Models shall be available at different scales, over time and space. For instance, there will be models with characteristic times ranging from hours to months. In the same way, some models will be valid only over a limited area of a few hectares or on a much wider region (e.g. NUTS2-3).

Use case 4 (Application developers): The research institute has access to their collection of SEAMLESS models, including models developed by other partners. They want to develop stand-alone applications for different kind of users: for end users, to use models to evaluate the impact of regional policies or technological innovations; for policy makers, to evaluate the impact of new laws and regulations. The requirements associated with this use case are:

| An environment for the development of workflow management | Must support a workflow development environment to create the application as a sequence of model transformations by tools, data manipulations and processing. This ensures that SEAMLESS allows the development of packaged customised applications. (see http://www.wfmc.org/) |
| Repositories for data, models, and tools | While the justification for the repository for models has been given in the previous use case, and we cannot even think of not having data repository (the question is how to implement it), tool repositories are a novelty. Tools must have the same status as models. Tools are the implementation of algorithms, they could be numerical integration routines, or a linear program solver, implementing the simplex algorithm. We need metadata for tools, to describe what they do, and how they are interfaced with models. |
| Allows web based deployment of tools | A tool providing a service (e.g. integration of a model) can be offered as a web service. The Semantic Web initiative is all about this. Links to the GRID computing initiative. Thanks to semantically rich interfaces, models become searchable in the webspace and an “agent” can manage a complex simulation coordinating the different models taking part in it. Possible links to the concept of “model federations” (see the DMSO HLA, High Level Architecture https://www.dmso.mil/public/transition/HLA) |
| Must support software quality checking | The quality of the software components must be testable, by using unit tests, integration test and regression tests. |

3.5 Other potential use scenarios for SEAMLESS for application and model development

1. A three-year project needs to build a crop rotation model by coupling 4 existing models - a crop model and a natural vegetation model both in Fortran, a water balance model in

---

2 This use case needs to be expanded to include those elements from the other use cases set out below which illustrate new requirements not already covered
Pascal, and a soil organic matter model developed in a graphical modelling environment with code export possibility to C++. The partners need to calibrate and validate their models using data from two years of field experiments (executed in the course of the project), stored in a common database. The project has meteorological data which are incomplete. Both the experimental data and the meteorological data will be checked and revised in the course of the project. Furthermore one project member doesn't trust the quality of the crop model, and needs to do some extensive testing, and perhaps further model development.

2. Build an application that lets users pick component models from a repository (the repository holds 10 crop models, 10 soil models, 10 aphid models, 10 meteo models, ..) and connect them to form ecosystem models.

3. Develop a cropping systems simulator (CSS) from components to simulate soil structure, soil water, soil C-N, annual and perennial crops, orchards, forestry, pesticides, diseases, weeds. Different participants independently develop different model components, according to their specific expertise. The simulator should allow the use of alternatives for model components e.g. developed by independent groups, and new types of component (e.g. an insects component). Need to handle state and time driven events. The CSS must work as a stand alone application, or driven (e.g. in many instances) by a higher level simulator (e.g. a Farm Activities Simulator). CSS performance will be tested against experimental field data, and scenarios will be evaluated (settings of resources, factors, and actions with specific or stochastic weather and on specific soils). Scenarios need to be validated before each run, and outputs from each run saved.

4. A Chinese professor collaborates with a SEAMLESS project, uses SEAMLESS to develop models, and develops a course in modeling at his university based on these models. No money to pay for a SEAMLESS license

5. (Current FP5 project) In a greenhouse climate control project one partner is developing an hydrological model using software under Linux. Another partner provides a crop growth model to allow climate control optimization based on crop performance, developed using FORTRAN in Windows. Does SEAMLESS offer a solution?

6. (Current FP5 project) Selection of structurally detailed statistically testable ecological models Objectives: This project will investigate the costs and benefits of models that provide a detailed description of ecological processes, using tools and example data sets available. The approach will be to select models, which are empirically justifiable in that they contain an amount of detail in their structure, which is appropriate to the data which are available. This appropriateness will be evaluated using formal statistical tests. By doing this it is intended to go as far as the information allows, but strictly no further.

### 3.6 Some more requirement issues

1. Portability (Linux/Windows). It has to be remarked that legacy models cannot usually be ported if they are provided in compiled form. They can be only invoked as “web services” from an application, but they must reside on their original environment.

2. Legacy models: one strong requirement was that SEAMLESS should support both open and closed source submodels and tools, to allow for use of legacy and proprietary models. Open source models are models where the source code is freely accessible and redistributable. Closed source models are those where the source code is either protected by an IPR agreement (even if the source code is accessible) or those where only binary code is available. Closed source components will inevitably be less portable.

3. Reusability of models/tools/data. Reusability is automatically achieved if we provide full access to the declarative modeling language we use to define our models.. It is the same methodology for preserving coding investment, covered above.
4. Interoperability. Maybe we also need a "Facility for Interoperability Testing" (FIT) aiding modellers/developers in delivering interoperable models, tools and services which can be used together. This may be covered by adequate meta-data. Checking the model and tools interfaces we can see whether models/tools/policies are interoperable.

5. The ability of using remote sensing data to calibrate and validate spatial models. This is related to the “quality issue”, but also to type of model. Within the meta-data set there must be information about this; whether it is an aggregated, point-base model, or a distributed. If we want to strictly address the issue of GIS models, there is a need for specific technical documentation. It may come in a later stage.
4 A comparison of existing modelling frameworks

During the meeting a list of 18 requirements emerged, which are listed and discussed in relation to the various suggested software solutions in Appendix 1. How should we prioritise these requirements?

Adopting the MoSCoW scheme (Must have, Should have, Could have, Would have) the requirements have been listed in descending order of number of votes received in the meeting. This voting highlighted the aspects thought to be important, but does not necessarily reflect the urgency with which the requirement should be or can be implemented.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>MSCW</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Uses declarative approach (formal representation of equations) for language independent model description (XML)</td>
<td>M-C 8</td>
<td>8</td>
</tr>
<tr>
<td>2 Knowledge-based support for modelling (e.g. advises users where coupling is appropriate)</td>
<td>M 8</td>
<td>8</td>
</tr>
<tr>
<td>3 Framework software is open source (e.g. GPL)</td>
<td>M 4</td>
<td>4</td>
</tr>
<tr>
<td>4 Allows for quantitative (static, dynamic, stochastic, deterministic, descriptive, explanatory…), qualitative and conceptual components</td>
<td>M 4</td>
<td>4</td>
</tr>
<tr>
<td>5 Supports both open and closed source submodels and tools</td>
<td>M 3</td>
<td>3</td>
</tr>
<tr>
<td>6 Version control for framework, components and scenarios</td>
<td>M 3</td>
<td>3</td>
</tr>
<tr>
<td>7 Facilitates self-documentation</td>
<td>M 2</td>
<td>2</td>
</tr>
<tr>
<td>8 Supports model and software quality checking</td>
<td>M 2</td>
<td>2</td>
</tr>
<tr>
<td>9 Allows web-based deployment of models and tools</td>
<td>C 2</td>
<td>2</td>
</tr>
<tr>
<td>10 Supports a variety of different users</td>
<td>M 2</td>
<td>2</td>
</tr>
<tr>
<td>11 Provides repositories for: a) data / models; b) solvers; c) knowledge</td>
<td>M 1</td>
<td>1</td>
</tr>
<tr>
<td>12 Provides a benchmarking environment</td>
<td>M 1</td>
<td>1</td>
</tr>
<tr>
<td>13 Portable between Linux and Windows</td>
<td>M 1</td>
<td>1</td>
</tr>
<tr>
<td>14 Allows development of packaged, customised applications</td>
<td>M 1</td>
<td>1</td>
</tr>
<tr>
<td>15 Reusable outside the framework</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>16 Can handle an evolving ontology (tag definitions)</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>17 Allows web-based publication of models and tools</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>18 Allows specialised interactions between components (e.g. by using a workflow environment)</td>
<td>M</td>
<td></td>
</tr>
</tbody>
</table>

On the basis of the above list of requirements, we selected four modelling frameworks (TIME, Modcom, IMA, OpenMI) and, with the help of the framework developers and supplemental material such as journal and conference papers, we checked how each one of the framework addressed the requirements.

This screening phase will be useful in extracting relevant design ideas and possibilities of software reuse in the design phase of SEAMFRAME.
4.1 TIME – The Invisible Modelling Environment

The Invisible Modelling Environment (TIME) is a new environmental modelling framework being developed within the Catchment Modelling Toolkit project in the CRC for Catchment Hydrology. TIME differs from existing modelling frameworks in a number of ways, particularly in its use of metadata to describe and manage models as well as the flexibility given to model developers to ‘pick and choose’ the components of TIME relevant for a given project. Functionality that is embedded as part of a monolithic core in other frameworks is included in applications under TIME on an as-needed basis using optional, interchangeable components. This flexibility extends to components that manage data and models, recognising that one approach does not necessarily fit all applications. TIME includes a number of small framelets supporting extension in key areas such as data representation and visualisation. All fundamental data types, such as rasters and time series, are defined within the data framelet, which supports the definition of new, compatible data types. The visualisation framelet allows the definition of ‘layers’, each providing a visual representation of some type of data, such as rasters or polygons. Multiple layers can be placed on a single ‘view’, such as overlaying a polygonal map on a raster. Views can be surrounded by ‘decorators’ such as axis and titles, each of which can be combined independently. TIME includes a number of tools, which operate generically on models, including an automatic user interface generator and various model optimisation tools. TIME is developed on the Microsoft .NET platform and supports the development of models in a variety of languages, including Visual Basic.NET, Fortran 95.NET, C# and Visual J#. TIME is currently being used to develop a range of modelling applications, including a library of rainfall runoff models and a model supporting assessment of stream ecosystem health under various flow scenarios.

4.1.1 Requirements matching

1. **Uses declarative approach (formal representation of equations) for language independent model description (XML)**

   No. The model description (such as input and output definition, variable naming), is handled through formal metadata specification. Given that TIME is implemented in .NET it supports multi-language development, but it is up to the modeller to adopt a programming style as close as possible to declarative modelling.

2. **Knowledge-based support for modelling (e.g. advises users where coupling is appropriate)**

   Not implemented, but the integral use of model metadata within the TIME environment supports the analysis of proposed model links through matching of data types. The lightweight architecture of the TIME core/kernel has the potential to support any knowledge-based model support system that can be expressed in model metadata.

3. **Framework software is open source (e.g. GPL)**

   The CRCCH (Co-operative Research Centre on Catchment Hydrology) has a federal government requirement to assess the commercial potential of any CRC products. However, it is the intention of the CRCCH to distribute as much as possible of the system under an open source agreement.

4. **Allows for quantitative (static, dynamic, stochastic, deterministic, descriptive, explanatory…), qualitative and conceptual components**
Yes. The TIME architecture has been designed to flexibly support a range of modelling methods and approaches. In line with CRCCH needs, the current models developments are almost entirely quantitative.

5. **Supports both open and closed source submodels and tools**

Yes. Through multi-language support and the capabilities of the .NET system, TIME supports linking with compatible open and closed sources models and tools. In addition, a proposed development in cooperation with the European Open Modelling Interface (OpenMI) ([www.harmonit.org](http://www.harmonit.org)) will increase the linking capabilities of TIME with a range of European hydraulic and hydrological models.

6. **Version control for framework, components and scenarios**

Framework and component development uses CVS for version control. No formal control of scenarios has been implemented.

7. **Facilitates self-documentation**

Yes. The metadata capabilities of both TIME and .NET support a reasonable level of self-documentation (assuming developers provide correct metadata and required fields when coding).

8. **Supports model and software quality checking**

No formal method for model and software quality checking has been constructed within the TIME environment.

9. **Allows web-based deployment of models and tools**

Yes. Models and tools will become progressively available from [www.toolkit.net.au](http://www.toolkit.net.au) over the next three years.

10. **Supports a variety of different users**

Yes. Five major user types have been considered during the requirements analysis.

11. **Provides repositories for: a) data / models; b) solvers; c) knowledge**

a) Yes. The toolkit website has been identified as the primary repository.

b) TIME supports pluggable solvers, but at present no specific solver repository is available within TIME. However, these would be handled similarly to a), above.

c) An information management system is under development, and a "toolkit assistant" that will act as the primary model knowledge system.

12. **Provides a benchmarking environment**

No. Benchmarking tools have been used for TIME development, but little development has been undertaken for model benchmarking.

13. **Portable between Linux and Windows**

.NET is currently Windows based. Open source development of the Linux equivalent (Mono [www.go-mono.com](http://www.go-mono.com)) offers potential for Linux deployment.

14. **Allows development of packaged, customised applications**

Yes. The component-based approach used with TIME supports development of tailored user interfaces for collections of components linked together to form specific modelling solutions.

15. **Models are reusable outside the framework**
Generally, TIME components (packaged as DLLs) are not reusable, unless the whole of TIME core is provided.

16. Can handle an evolving ontology (tag definitions)

Not at present, but it will be possible through evolution of TIME metadata definition and utilisation.

17. Allows web-based publication of models and tools

Yes. See (9) above.

18. Allows specialised interactions between components (e.g. by using a workflow environment)

TIME supports the development of tailored components interactions through metadata manipulation.

4.1.2 Potential contribution to SEAMLESS

TIME is the result of many years of research and development by Australia’s Co-operative Research Centre for Catchment Hydrology. Despite being focused on catchment modelling, the underlying ideas make it a source of inspiration for SEAMLESS. The fact of not being open-source hinders the reuse of the software code. Had it been possible, a number of data manipulation and visualization tools would have been available to SEAMLESS. The smart use of introspection (the ability to discover the software property within the program itself) will make it possible to develop SEAMLESS software components to be re-used in TIME and possibly vice-versa.

4.1.3 Contact details

Robert Argent (Project Leader)       R.Argent@unimelb.edu.au
Joel Rahman (Software Engineer)   Joel.Rahman@csiro.au
Cooperative Research Centre for Catchment Hydrology (CRCCH)
Hyperlink: www.catchment.crc.org.au
4.2 IMA: the Integrating Modelling Architecture

The IMA derives its power and generality from adopting the semantics of the natural entities represented (e.g. economic value, biomass, or nitrogen flow) as opposed to that of the execution workflow that calculates the desired results. The most general representation of a modeled entity in the IMA is a tree of interconnected modules, each corresponding to a precisely identified object of study. The declaration of each module can include an observation context that encompasses all aspects connected to scaling, such as granularity and extent in space and time, and can be manipulated by the investigator or by the system when mediating across different, compatible representations used together.

4.2.1 Requirements matching

1. Uses declarative approach (formal representation of equations) for language-independent model description (XML).

Yes. The IMA is a modular, extendible object system where both classes and objects are specified in XML. XML-specified modules can represent data, models, optimisers, algorithms.

2. Knowledge-based support for modelling (e.g. advises users where coupling is appropriate).

Yes. The IMA’s primary goal is to give modellers the ability of using the bare conceptual bones of the problems while promoting a design discipline that automatically enforces constraints of semantics, space, time, and other applicable domains (e.g. consistency in measurement protocols or bibliographic source) - as well as the obvious constraints of storage type and units. Modules are tagged with semantic types (pointers into formally specified ontologies) and contain domain objects that represent cross-cutting aspects related to observation (space, time). Compatibility between all these is automatically enforced by specialised interfaces built over a common API. Domain types and ontologies can be added to the system in a plug-in fashion. The permanent storage functionalities and database interfaces have the ability of retrieving only semantics- and domain-compatible objects that will fit a precise role in an existing model structure. The domain functionalities are developed in the core engine and under implementation in the time and space domains. The semantic type functionalities are under development.

3. Framework software is open source (e.g. GPL).

Yes. Everything in the IMA is covered by the GPL, and there is no functionality that requires the use of closed-source software. This has been a main design goal for the IMA. All extensions use open source software (Mapserver/GEOS for GIS, R for statistics, PostgreSQL for permanent storage) although they can be implemented with commercial solutions if required.

4. Allows for quantitative (static, dynamic, stochastic, deterministic, descriptive, explanatory…) qualitative and conceptual components.

The IMA supports multiple modelling paradigms and is not by any means limited to equations as a way to describe a model. For modules whose specification requires equations (e.g. those whose state is calculated by integrating a differential equation), these can be specified entirely in XML if desired, although it is normally more intuitive to use some sort of expression of programming language. The IMA supports a full-faced, object-oriented programming language and has plug-in support for other languages (Scheme, Javascript, Python, C/C++). Currently the modules implementing difference equation modelling, the STELLA importer (and probably the Simile importer...
that will be developed) use CDATA sections and the internal IMA language compiler rather than MathML or other XML-based ways to express the equations. Supporting MathML if required would require very little effort.

5. **Supports both open and closed source submodels and tools.**
Yes. Everything in the IMA can be plugged in as an extension and all software with a C API can be used. One specialised class of objects handles interaction with closed executables, through proxy objects that range from command line drivers to CORBA/COM peers for enabled components. This allows legacy programs to be given the necessary semantic characterisations and wrapped at the executable level, becoming a component of a larger-scale model implemented in the IMA.

6. **Version control for framework, components and scenarios.**
Under development. The permanent storage interface allows version control and specialised, XML-specified “difference” objects are planned that can be applied to other objects to modify them. This is a central feature to enable a “bulletin-board” approach where the objects of discussion are actual data and models.

7. **Facilitates self-documentation.**
Yes. The core IMA engine has a template-based self-documentation system that uses templates to enable self-documentation of objects in formats such as HTML, text, or LaTeX. This system is currently used for all web-enabled database applications based on the IMA. Plug-in support for MIME types can be also plugged in on a class-specific basis. This feature is used to allow objects to intelligently select their proper multimedia representation given a MIME type (e.g. spatial objects will generate GIS maps, temporal objects will generate timeseries graphs, and spatio-temporal objects will generate animated maps, all as an answer to the same request for a image/png).

8. **Supports model and software quality checking.**
Under development. Storing accuracy information along with data and model is a crucial feature in an environment that allows arbitrary composition of data sources and processing algorithms (with associated compounding of error) and allows automatic rescaling of differently scaled information (with associated transformation error). Such features are not implemented in the IMA at this time.

9. **Allows web-based deployment of models and tools.**
Yes. This is a core requirement for all the projects that the IMA is currently employed in. A specialised IMA runtime runs as a multi-user server and a CGI IMA client allows full control of an IMA object database and hierarchy through the web. The self-documentation engine does the rest, allowing complete deployment of models, data, and analysis tools.

10. **Supports a variety of different users.**
Yes. IMA-enabled applications can be used as simple databases from the Web while modellers can use the IMA as an XML modelling tool and runtime environment. Different runtimes can be created very simply using policy-based design, supporting batch modelling, client/server interaction, or GUI interaction (currently not implemented).

11. **Provides repositories for a) data/models; b) solvers; c) knowledge.**
The IMA runtime has a virtual repository interface that uses XML-specified views and queries. It has been currently implemented on top of PostgreSQL out of consideration of stability, open source and availability of specialised spatial operator. All of the three
classes of objects mentioned in the requirement map into representational categories handled by the IMA and its permanent storage system.

12. Provides a benchmarking environment.

   Not available. In general the explicit semantics of IMA modules allows many metrics to be calculated on them (e.g. the runtime system can estimate the resource requirements of a model by its representation, and enforce hard constraints on it before trying to allocate and run it).

13. Portable between Unix and Windows.

   Yes. The IMA is written in ANSI C++ and has no dependence on particular OS architectures. All the development so far has been done on Linux.


   Under development. IMA models can be executed by a runtime and it’s easy to package a model with its own runtime system. A feature in development is the translation of a whole model into C language for compilation and delivering as an executable.

15. Reusable outside the framework.

   Yes. IMA modules are executed by a runtime that can assume many different forms and be hosted on a local computer or a server. Policy-based software design allows the runtime operation to be redefined in a very simple way.

16. Can handle an evolving ontology (tag definition).

   Under development. IMA modules can be tagged with semantic types that point to specific ontologies. The issue of tracking evolving ontologies without losing semantic integrity is, anyway, extremely tricky. The SEEK project (http://seek.ecoinformatics.org) has a Semantic Mediation and a Knowledge Representation working group that is discussing how to deal efficiently with evolving semantics.

17. Allows web-based publication of models and tools.

   Yes. the model repository can be easily accessed from the web.

18. Allows specialised interactions between components (e.g. by using a workflow environment).

   Yes. The IMA is a workflow environment by its own nature, and it has close relationships to Kepler (http://kepler.ecoinformatics.org/) and Ptolemy II (http://ptolemy.eecs.berkeley.edu/ptolemyII/).

4.2.2 Potential contribution to SEAMLESS

IMA is a design, and parts of it have been implemented in IMT (Integrated Modelling Toolkit). IMA and IMT are highly relevant to SEAMLESS since many of the fundamental requirements are covered by IMA. A first assessment sees in the declarative modeling capabilities of IMA the most interesting features, together with the ability to combine tools and models in workflows. Finally, the knowledge base of SEAMLESS would benefit from being inspired by IMA.

4.2.3 Contact details

Ferdinando Villa, Project Leader (ferdinando.villa@uvm.edu)
Ecoinformatics Collaboratory, Gund Institute for Ecological Economics
University of Vermont, USA

4.3 Modcom

MODCOM’s central concept is that of well-defined, self-describing component models, either written specifically for MODCOM or included through an adapter. A MODCOM simulation is constructed as a collection of interconnected components. MODCOM’s component architecture provides a robust base for building flexible, domain-specific tools for such tasks as visual construction of components and simulations, specification of spatial relationships, and optimisation.

![Diagram](Image)

*Figure 1. Proposed Simulation and Collaboration Architecture*

The MODCOM core functionality has been implemented. A number of agro-ecological process components have been developed. Two tools for visual construction of simulation models have been recently developed.

4.3.1 Requirement matching

1. **Uses declarative approach (formal representation of equations) for language independent model description (XML)**
   
   No. MODCOM components are defined at the binary level; such components can be written in a declarative language just as well as in Fortran, C/C++ or Java. It is anyway possible to automatically convert a Simile declarative model into a MODCOM component.

2. **Knowledge-based support for modelling (e.g. advises users where coupling is appropriate)**
   
   Not available.

3. **Framework software is open source (e.g. GPL)**
   
   Yes. The MODCOM source code is fully available.

4. **Allows for quantitative (static, dynamic, stochastic, deterministic, descriptive, explanatory…), qualitative and conceptual components**
MODCOM allows quantitative components. Extensions might be required for qualitative components.

5. **Supports both open and closed source submodels and tools**

MODCOM is written in C++ on the Microsoft COM platform and a .NET version is under development. The source code of a model, written in C# or in C++ can be made explicit, or packaged in a binary component.

6. **Version control for framework, components and scenarios**

The MODCOM framework is available under concurrent version control (CVS). Models can also be distributed under the versioning system, but there are no facilities for data and scenario handling.

7. **Facilitates self-documentation**

MODCOM components are defined at the binary level; MODCOM wants components to do their job but doesn’t care how they do it. A component could make available the specification of the model it represents and documentation could be generated from that.

8. **Supports model and software quality checking**

A component could provide information about valid ranges for inputs; a checking component could be written to signal when those ranges are exceeded. A component could make available the specification of the model it represents; checks could be made on the basis of it.

9. **Allows web-based deployment of models and tools**

Currently no facilities, but MODCOM-based components and tools could be deployed through the Web.

10. **Supports a variety of different users**

Presently the model coder, who writes source code, and the model runner.

11. **Provides repositories for: a) data / models; b) solvers; c) knowledge**

Not currently. These tools are envisioned in a future release.

12. **Provides a benchmarking environment**

Not currently. Envisioned in a future release.

13. **Portable between Linux and Windows**

MODCOM is currently under porting to the .NET environment, for which a runtime environment is also available on Linux.

14. **Allows development of packaged, customised applications**

Yes. Assembling models and components allows to create specialised applications, targeted for a particular usage.

15. **Reusable outside the framework**

MODCOM models fully depend on the Modcom framework, in the same way as TIME models.

16. **Can handle an evolving ontology (tag definitions)**

Not currently supported.

17. **Allows web-based publication of models and tools**
Currently no facilities, but MODCOM-based components and tools could be deployed through the Web.

**18. Allows specialised interactions between components (e.g. by using a workflow environment)**

Yes, but it is up to the programmer to define the flow of the logic in the source code of an application which uses the MODCOM framework.

### 4.3.2 Potential contribution to SEAMLESS

MODCOM has many contact points with TIME. It can be considered a modeling and simulation framework and it has the advantage of having been thought for agro-ecological models from the beginning. For instance, MODCOM can combine discrete and continuous simulation paradigms to handle management events during biophysical simulations.

MODCOM is also open-source and this grants for its full re-use within SEAMLESS.

On the down side, MODCOM has not a fully developed suite of applications (or framelets) as TIME and therefore adapting it for use to simulate biophysical models will be more labour intensive.

### 4.3.3 Contact details

Frits van Evert ([frits.vanevert@wur.nl](mailto:frits.vanevert@wur.nl))

Plant Research International

Wageningen University, NL

Hyperlink: [http://www.modcom.wur.nl](http://www.modcom.wur.nl)
4.4 OpenMI

OpenMI stands for Open Modelling Interface and Environment, a standard for model linkage in the water domain. OpenMI is born as a product of the HarmonIT project, which aims at providing an infrastructure to unify and link models and tools for catchment management at a European scale. OpenMI consists of a set of software interfaces that allow new and existing models to interact with each other, with sources of data and with instruments and tools for the display and analysis of data.

In OpenMI a model can be regarded as an entity that can provide and/or accept data. OpenMI is based on direct access of the model at runtime, thus not using files for data exchange. A model is therefore implemented as a component, which can be accessed through a standard interface.

Parts of this review are based on the official OpenMI documentation, on an internal document provided by Alterra (Critical assessment of OpenMI), and on a Review of OpenMI architecture reports A & B, by Dr Hamish Harvey (http://www.cen.bris.ac.uk/pgra/dph/publications/2003-05-28-openmi1.pdf).

4.4.1 Requirements matching

1. Uses declarative approach (formal representation of equations) for language independent model description (XML)
   No. OpenMI is aimed at connecting existing models, which are ‘black-boxes’ wrapped up in a standard interface.

2. Knowledge-based support for modelling (e.g. advises users where coupling is appropriate)
   Not implemented. Matching model interfaces and model composition is greatly enhanced by adopting the common OpenMI interface, but model teleology and semantics are not presently supported.

3. Framework software is open source (e.g. GPL)
   The OpenMI software is supposed to be distributed as open-source.

4. Allows for quantitative (static, dynamic, stochastic, deterministic, descriptive, explanatory…), qualitative and conceptual components
   The OpenMI interface is neutral with respect to the adopted modelling paradigm, since it concentrates on the data exchange. Yet, values are crisp and they are quantitative.

5. Supports both open and closed source submodels and tools
   Yes. A model can be an opaque black-box, which simply implements the OpenMI interface, but a model can also be structured according to a ‘template’ provided by a class, which has methods such as initialise, and get_values, that contain the model code. The modeller is free to customise these methods and develop his/her own model implementation.

6. Version control for framework, components and scenarios
   Not implemented.

7. Facilitates self-documentation
   Yes. Each model has a description and an ID to facilitate model archival and retrieval.

8. Supports model and software quality checking

Page 34 of 49
No formal method for model and software quality checking has been constructed within the OpenMI environment.

9. **Allows web-based deployment of models and tools**

This wasn’t the focus of the development, but, being models implemented as components, it is easy to transform them into web services.

10. **Supports a variety of different users**

It supports model developers and model linkers. Finished applications, based on the OpenMI architecture, can be targeted to the end-users.

11. **Provides repositories for: a) data / models; b) solvers; c) knowledge**

While modelling knowledge is not explicitly supported, models can be stored for later usage, and the same holds for data visualisation and analysis tools.

12. **Provides a benchmarking environment**

Not available.

13. **Portable between Linux and Windows**

OpenMI is available in C# (runs on the .NET framework) and in Java, which runs on multiple platforms.

14. **Allows development of packaged, customised applications**

Yes. The component-based approach allows to package applications linking sub-models.

15. **Models are reusable outside the framework**

OpenMI reuses models provided by other frameworks. A model implementing the OpenMI interface can be reused by another framework which is compliant with that interface.

16. **Can handle an evolving ontology (tag definitions)**

The semantic annotation of model inputs and outputs is not available. Still there are three interfaces to describe what’s returned from an OpenMI model IQuantity/IUnit/IDimension. OpenMI provides a warning mechanism that kicks in when things are obviously wrong. The terms “unit” and “dimension” are being used loosely.

17. **Allows web-based publication of models and tools**

Not explicitly, but it is easy to implement.

18. **Allows specialised interactions between components (e.g. by using a workflow environment)**

The OpenMI pull architecture favours model linking and reuse and components can be combined in a variety of ways. The OpenMI pull-model is expected to impose the least amount of restrictions in linking models: quantities are ‘pulled’ from models, which ask other models to provide what they need.

### 4.4.2 Potential contribution to SEAMLESS

OpenMI, as TIME, was originally intended for integrating catchment hydrology models. The basic principle behind OpenMI is that hydrological models are big, complex applications, and we therefore need a software framework to handle the hand over of data from application to application, not limiting to a simple pipeline, but also allowing for feedback loops. While OpenMI remains a framework for application linking, it also becomes very interesting to
provide a layer for integration of different SEAMLESS applications at various levels of scale, such as Economic Models at the European scale, with Farm-Economic models at the regional scale, and biophysical models at the field scale.

4.4.3 Contact details

Hyperlink: www.openmi.org
References


Gamma, E, Helm, R. Johnson, R., Vlissides, J. 1995. Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley, Boston.


Glossary

**Application**: it is a software package obtained from the modelling environment. An application includes a custom GUI and it includes several components.

**Architecture**: Blue-print and styles to define structure;

**Component**: it is a model of a physical piece of the system being built. For example, source code files, DLL's, Java beans and other discrete pieces of the system may be represented as components. By building the system in discrete components, localisation of data and behaviour allows for decreased dependency between classes and objects, providing a more robust and maintainable design. Components may be either models or tools/utilities.

**Declarative code**: computer code where the order of the statements is not relevant for its execution. Declarative code contains the model equations, but it does not tell how to compute a result using the equations. Modelling is an activity well suited to be described with declarative code.

**Imperative code**: computer code that explains how to solve a problem, following a unambiguous and definite sequence of steps. Imperative code processes declarative code. Simulation is an activity which requires imperative code.

**Integrated framework**: an application which allows the evaluation of agricultural systems accounting for technical, environmental, economic and social indicators. One or more integrated frameworks will be the main deliverables of the integrated project.

**Model**: focused simplification of (a phenomena or process in) the real world;

**Modelling environment**: a software which allows developing components and applications. The modelling environments contains components which include a GUI for either component or application development.

**Legacy model**: a model, often packaged in an application, inherited from someone else.

**Modelling framework**: the kernel component for static and dynamic model components use.

**Ontology**: it is a specification of a conceptualisation. Once a modeling exercise has defined all the variables and relationships existing in a given model, this information can be stored in an ontology. There are many languages which can be used to represent an ontology, but RDFS/OWL is one of the most common since, being based on XML, it can be easily processed by computers.

**Project**: when referred in the use cases of the user Application/component developer (see below) a project is either a component or a system model which includes several components. When the user is a farmer or a policy maker, a project is the use of an application in specific conditions (a farm, a region etc.)

**Programming interface**: a set of component interfaces, packaged in a binary object together with their implementations, with the necessary documentation to reuse them in a software application.

**Proprietary model**: a model for which the rights to access, use and inspect its code are limited by Intellectual Property Rights.

**Simulation**: monitoring dynamics of attributes (in time and/or in space);

**Software framework**: productivity tool(box) to assemble components using architecture
Open Source: software code for which you have complete access to the source code. Moreover, the source is freely usable and re-distributable under a license agreements such as GPL (Gnu Public License), LGPL and so on;

System: a portion of the real world, with clearly defined borders, described both in a static and dynamic fashion by models
Appendix 1. SEAMLESS requirements and suggested solutions

<table>
<thead>
<tr>
<th>Requirement</th>
<th>M</th>
<th>E</th>
<th>C</th>
<th>Notes/Motivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Discussion of SIMILE model structure and XML format.</td>
</tr>
<tr>
<td>2</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Discussion of SIMILE model structure and XML format.</td>
</tr>
</tbody>
</table>

Yes, FIW has been developed specifically as a proof-of-concept demonstrator for a declarative modelling approach. Models are saved in an open text format: currently Prolog, we have a prototype converter to and from XML.

Yes / no. FIW is not explicitly build to support language independent model descriptions. You can however use wrapper around models that do.

Python provides tools for meta-modelling (Epilogue, Bash).

Yes / no. GP uses XML to specify model meta-data. In this specification, only simple meta-data is provided to support the registration of the model within the GF.

Our GCF (general coupling framework), see http://www.cs.man.ac.uk/cnc/bin/cnc_gcf.pl, supports and promotes the flexible composition and deployment of coupled models. It uses XML to capture variables describing (CVP-complaint see below) models which are to be composed together, composition information describing how models will be coupled (in terms of their input and output requirements) and information regarding the required deployment on (an appropriate set of) computational resources and communication mechanisms.

Table of requirements and suggested solutions.
| Requirement | M | L | W | F | U | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|-------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Simile      |   |   |   |   |   |   |   |   | Yes | No | No | No | No | No | No | No |
| FIW         |   |   |   |   |   |   |   |   | Yes | No | No | No | No | No | No | No |
| Python      |   |   |   |   |   |   |   |   | Yes | No | No | No | No | No | No | No |
| GCF         |   |   |   |   |   |   |   |   | Yes | No | No | No | No | No | No | No |
| IMA         |   |   |   |   |   |   |   |   | Yes | No | No | No | No | No | No | No |
| TIME        |   |   |   |   |   |   |   |   | Yes | No | No | No | No | No | No | No |
| MODCOM      |   |   |   |   |   |   |   |   | Yes | No | No | No | No | No | No | No |

### Framework software

| Requirement | M | L | W | F | U | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|-------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Simile      |   |   |   |   |   |   |   |   | Yes | No | No | No | No | No | No | No |
| FIW         |   |   |   |   |   |   |   |   | Yes | No | No | No | No | No | No | No |
| Python      |   |   |   |   |   |   |   |   | Yes | No | No | No | No | No | No | No |
| GCF         |   |   |   |   |   |   |   |   | Yes | No | No | No | No | No | No | No |
| IMA         |   |   |   |   |   |   |   |   | Yes | No | No | No | No | No | No | No |
| TIME        |   |   |   |   |   |   |   |   | Yes | No | No | No | No | No | No | No |
| MODCOM      |   |   |   |   |   |   |   |   | Yes | No | No | No | No | No | No | No |

### Framework software

| Requirement | M | L | W | F | U | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|-------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Simile      |   |   |   |   |   |   |   |   | Yes | No | No | No | No | No | No | No |
| FIW         |   |   |   |   |   |   |   |   | Yes | No | No | No | No | No | No | No |
| Python      |   |   |   |   |   |   |   |   | Yes | No | No | No | No | No | No | No |
| GCF         |   |   |   |   |   |   |   |   | Yes | No | No | No | No | No | No | No |
| IMA         |   |   |   |   |   |   |   |   | Yes | No | No | No | No | No | No | No |
| TIME        |   |   |   |   |   |   |   |   | Yes | No | No | No | No | No | No | No |
| MODCOM      |   |   |   |   |   |   |   |   | Yes | No | No | No | No | No | No | No |

### Framework software

| Requirement | M | L | W | F | U | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|-------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Simile      |   |   |   |   |   |   |   |   | Yes | No | No | No | No | No | No | No |
| FIW         |   |   |   |   |   |   |   |   | Yes | No | No | No | No | No | No | No |
| Python      |   |   |   |   |   |   |   |   | Yes | No | No | No | No | No | No | No |
| GCF         |   |   |   |   |   |   |   |   | Yes | No | No | No | No | No | No | No |
| IMA         |   |   |   |   |   |   |   |   | Yes | No | No | No | No | No | No | No |
| TIME        |   |   |   |   |   |   |   |   | Yes | No | No | No | No | No | No | No |
| MODCOM      |   |   |   |   |   |   |   |   | Yes | No | No | No | No | No | No | No |
5 Supports both open and closed submodels and tools M 3 2 Since Simile is a representative proof-of-concept demonstrator for declarative modelling, test is the natural format for representing models. This is intrinsically open-source, both for viewing and for adapting. No doubt it is possible to engineer particular functions (such as digital signatures) for particular software tools, but these would be specific to those tools. It would also be possible to have a declarative modelling approach that supports binary/executable files, but it would not, I think, get much support from tool developers and is contrary to the spirit of declarative modelling. Tools (input/output, display, net control) are written in TCL/Tk, an interpreted language. They are thus open-source-readable. Currently, they are not licensed for open-source-adapting. There is no restriction on any Simile user developing their own tools.

6 Version control for framework, components and scenarios M 3 1 All aspects of Simile are suitable for version control. We make extensive use of CVS in Simulistics.)

7 Facilitates self-documentation M 2 3 This is a major strength of Simile. The combination of an ontology.

6 Version control for framework, components and scenarios M 3 1 No. The framework itself and the applications and components are stored in a Visual SourceSafe database. Support is managed by the company MAN/Systèmes in Rijswijk (NL).

7 Facilitates self-documentation M 2 3 No. Not supported. In the standard Python library, there are several tools to generate documentation about.

5 Supports both open and closed submodels and tools M 3 2 Yes. Models can be delivered as a dynamic link library or executable, or directly implemented in an FIW application. Yes. Model applications can be provide in DLL or EXE. For each application a wrapper has to be written to make the application available to the framework. Yes. Python is freely usable and distributable, even for commercial use. Components are loaded dynamically so closed source submodels can be used with open source one with respect of open source license requirement.

4.5.2c This is unacceptable, we have techniques which enable models to be ‘wrapped’ in a transformation wrapper which converts a models output to the required (put) and (get) model (dealing with any control issues – relating to whether the data exchange mechanism is a ‘push’ or ‘pull’ model). These have relatively simple control requirements. Models are converted using a shell-like approach. We are currently investigating handling more complex control structures.

5.5.2c Yes. Yes. Through mutual-language support and the capabilities of the .NET system, TIME supports linking with compatible open and closed source modules and tools. In addition, a proposal development in cooperation with the European Open Modelling Interface (OpenMI) (www.harmoni.org) will increase the linking capabilities of TIME with a range of European hydraulic and hydrological models.

5.5.3c Yes. The framework generator software (Concurrent Version System) is a widely used system for open source software. It is easy to use (lists of CVS features) and support many developers. Yes. CVS (Concurrent version system) is a widely used system for open source software. It is easy to use (lists of CVS features) and support many developers.

5.5.4c Support is managed by the company MX-Systems in Rijswijk (NL). No. The framework itself and all software tools (input/output, display, control) are written in Tcl/Tk, an interpreted language. They are thus open-source-readable. Currently, they are not licensed for open-source-adapting. There is no restriction on any Simile user developing their own tools.

5.5.5c Yes. Yes.  Through multi-language support and the capabilities of the .NET system, TIME supports linking with compatible open and closed source modules and tools. In addition, a proposal development in cooperation with the European Open Modelling Interface (OpenMI) (www.harmoni.org) will increase the linking capabilities of TIME with a range of European hydraulic and hydrological models.

5.5.6c This is intrusive, we have techniques which enable models to be ‘wrapped’ in a transformation wrapper which converts a models output to the required (put) and (get) model (dealing with any control issues – relating to whether the data exchange mechanism is a ‘push’ or ‘pull’ model). These have relatively simple control requirements. Models are converted using a shell-like approach. We are currently investigating handling more complex control structures.

5.5.7c No. Models can be delivered as a dynamic link library or executable, or directly implemented in an FIW application.

5.5.8c Yes. Models can be delivered as a dynamic link library or executable, or directly implemented in an FIW application. Yes. Model applications can be provide in DLL or EXE. For each application a wrapper has to be written to make the application available to the framework.

5.5.9c Yes. Python is freely usable and distributable, even for commercial use. Components are loaded dynamically so closed source submodels can be used with open source one with respect of open source license requirement.

5.5.10c Yes. Yes.  Through multi-language support and the capabilities of the .NET system, TIME supports linking with compatible open and closed source modules and tools. In addition, a proposal development in cooperation with the European Open Modelling Interface (OpenMI) (www.harmoni.org) will increase the linking capabilities of TIME with a range of European hydraulic and hydrological models.

5.5.11c Yes. Yes.  Through multi-language support and the capabilities of the .NET system, TIME supports linking with compatible open and closed source modules and tools. In addition, a proposal development in cooperation with the European Open Modelling Interface (OpenMI) (www.harmoni.org) will increase the linking capabilities of TIME with a range of European hydraulic and hydrological models.
<table>
<thead>
<tr>
<th>Requirement</th>
<th>M</th>
<th>L</th>
<th>C</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simile</td>
<td>M</td>
<td>L</td>
<td>C</td>
<td>R</td>
</tr>
<tr>
<td>FIW</td>
<td>M</td>
<td>L</td>
<td>C</td>
<td>R</td>
</tr>
<tr>
<td>Python</td>
<td>M</td>
<td>L</td>
<td>C</td>
<td>R</td>
</tr>
<tr>
<td>GCF</td>
<td>M</td>
<td>L</td>
<td>C</td>
<td>R</td>
</tr>
<tr>
<td>MODCOM</td>
<td>M</td>
<td>L</td>
<td>C</td>
<td>R</td>
</tr>
</tbody>
</table>

**Simile**
- Framework for Integrated Water Management (but broader than water, so name will change soon)
- christophe.pradal@cirad.fr
- http://biosys.bre.orst.edu/modcom/index.htm

**FIW**
- General Framework
tony@inria.fr
- www.inria.fr/sem/sem-pub/finmod.htm

**Python**
- chrisrleigh@us.ibm.com
- http://www.python.org

**GCF**
- CNC General Coupling Framework
- Frits.vanevert@wur.nl

**MODCOM**
- It's separate from GCF
- http://www.wur.nl

**TIME**
- The Invisible Modelling Environment
- R. Argon/atm.iue.ac.be
- http://www.wur.nl

<table>
<thead>
<tr>
<th>Requirement</th>
<th>M</th>
<th>L</th>
<th>C</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Supports model and software-quality checking</td>
<td>M</td>
<td>L</td>
<td>C</td>
</tr>
<tr>
<td>9</td>
<td>Allows web-based deployment of models and tools</td>
<td>M</td>
<td>L</td>
<td>C</td>
</tr>
<tr>
<td>10</td>
<td>Supports a variety of different users</td>
<td>M</td>
<td>L</td>
<td>C</td>
</tr>
</tbody>
</table>
Yes. The toolkit website has been identified as our primary repository.

Layer 4 of the diagram above. Different runtimes can be created very simply using policy-based design, supporting batch modelling, distributed interaction, or GUI interaction (currently not implemented).

We have benchmarking tools for TIME development, but little development has been undertaken for model benchmarking.
### Requirement | M | D | P | U | F
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Simile</td>
<td>Architecture is all supported. Active development for Apple OS X is underway (in beta state).</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIW</td>
<td>Framework for Integrated Water management (but broader than water, so name will change soon)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GF</td>
<td>General Framework</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Python</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GCF</td>
<td>CNC General Coupling Framework</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMA</td>
<td>Integrating Modelling Architecture</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODCOM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td>The Invisible Modelling Environment</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### References

- [monodex.html](http://biosys.bre.orst.edu/modcom/index.htm)
- [http://www.sf.net/projects/imt](http://www.sf.net/projects/imt)
- [http://www.cs.man.ac.uk/cnc-bin/cnc_gcf.pl](http://www.cs.man.ac.uk/cnc-bin/cnc_gcf.pl)
- [http://www棠catchment.crc.org.au](http://www棠catchment.crc.org.au)
- [Griley@cs.man.ac.uk](http://www棠catchment.crc.org.au)
- [Tamme.vanderwal@wur.nl](http://www棠catchment.crc.org.au)
- [R.Warren@uea.ac.uk](http://www棠catchment.crc.org.au)
- [R.Argent@unimelb.edu.au](http://www棠catchment.crc.org.au)
- [christophe.pradal@cirad.fr](http://www棠catchment.crc.org.au)
- [Frits.vanevert@wur.nl](http://www棠catchment.crc.org.au)
- [tammy.vanderwal@wur.nl](http://www棠catchment.crc.org.au)

#### Table:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>M</th>
<th>D</th>
<th>P</th>
<th>U</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>14 Active development of packaged, customised applications</td>
<td>2 Yes</td>
<td>2 Yes</td>
<td>10 above</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15 Reusable outside the framework</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>16 Can handle an evolving ontology (tag definitions)</td>
<td>3</td>
<td>2</td>
<td>1 The Simile XML Schema is currently rather fluid (the Prolog format is still Simile’s native model representation format). As a result, we plan to introduce namespace (at least two: one for System Dynamics elements, one for the object-based aspects, possibly using XMI). Additionally, MathML might be used for mathematical expressions, but this will considerably increase the size of file. However, even if there were only a single Simile Schema, it would still be possible for the Schema to be extended, or for a</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
### Requirement - Deliverable number: PD 5.2.2

**Group (e.g. SEAMLESS) to add their own namespace.**

1. Atlassian table-based publication of models and tools
   - **MICH**
   - **2**
   - **Y**
   - **P**
   - **T**

   - **Yes. Models will be published in XML documents. These will be viewable through a variety of XML Tools. On as documents generated by XSLT from the raw XML.**
   - **No.**
   - **No.**
   - **Yes.**
   - **Yes.**
   - **Yes.**
   - **See above.**
   - **See above.**
   - **See above.**
   - **See above.**
   - **See above.**

### Additional comments

**Declarative modelling can potentially be integrated with component-based modelling in 3 main ways.**

1. **Free, a declarative modelling tool like Simile can be used to produce individual components.**
   - **Second, a declarative modelling environment can include programmatic elements in XML that are referred to in effect what the Tyndall Centre's integrating framework does, with XML wrappers for each component, and the complete model specified as an XML document.**

   **Then, component-based architectures can provide components for many support tasks (links to GIS, running simulations, etc.), with the model itself being a single component (either programmed or produced in a declarative modelling environment).**

   **Therefore, it is possible to envisage ways of combining a declarative approach with a component-based approach.**

**However, even if both the first and second methods are used in the previous point (i.e. components are specified declaratively, and the component structure of the model is represented declaratively), these are**

**FIW** is a programming tool used for designing integrated applications and producing source code. A Delphi library is used to manage and link components. A graphical interface allows users to connect models and define data paths.

**Python** is an interpreted, interactive, object-oriented programming language. It combines functionality, power, and flexibility. Python supports a wide range of applications. It is an end-user environment that can be used to design and create models. A graphical interface allows users to connect models and define data paths.

**Simile** is a tool for building declarative models. It is a free and open-source tool that allows users to create models and simulate their behavior.

**GF** is a wide range of programming languages. It is a design tool for designing integrated applications and producing source code. A Delphi library is used to manage and link components. A graphical interface allows users to connect models and define data paths.

**GF** is a design tool for designing integrated applications and producing source code. A Delphi library is used to manage and link components. A graphical interface allows users to connect models and define data paths.

- **frits.vanevert@wur.nl**
- **Tonny.Otjens@wur.nl**
- **ferdinando.villa@uvm.edu**
- **University of Vermont**
- **http://biosys.bre.orst.edu/modcom/index.htm**

**IMPA** is a tool for building declarative models. It is a free and open-source tool that allows users to create models and simulate their behavior.

**Griley@cs.man.ac.uk**

- **http://www.ierm.ed.ac.uk/simile/in**

**MODCOM** is a component-based architecture for building flexible, domain-specific tools for such tasks as building and simulating integrated models.

**TIME** is a component-based architecture for building flexible, domain-specific tools for such tasks as building and simulating integrated models.

**Tyndall centre**

- **Terry.Otgian@weir.com**
- **Tsemann@vanderbilt.edu**

**Python** is an interpreted, interactive, object-oriented programming language. It combines functionality, power, and flexibility. Python supports a wide range of applications. It is an end-user environment that can be used to design and create models. A graphical interface allows users to connect models and define data paths.

**Pyhton** is an interpreted, interactive, object-oriented programming language. It combines functionality, power, and flexibility. Python supports a wide range of applications. It is an end-user environment that can be used to design and create models. A graphical interface allows users to connect models and define data paths.

**Pyhton** is an interpreted, interactive, object-oriented programming language. It combines functionality, power, and flexibility. Python supports a wide range of applications. It is an end-user environment that can be used to design and create models. A graphical interface allows users to connect models and define data paths.

**Pyhton** is an interpreted, interactive, object-oriented programming language. It combines functionality, power, and flexibility. Python supports a wide range of applications. It is an end-user environment that can be used to design and create models. A graphical interface allows users to connect models and define data paths.

- **Pyhton** is an interpreted, interactive, object-oriented programming language. It combines functionality, power, and flexibility. Python supports a wide range of applications.

- **Pyhton** is an interpreted, interactive, object-oriented programming language. It combines functionality, power, and flexibility. Python supports a wide range of applications.

- **Pyhton** is an interpreted, interactive, object-oriented programming language. It combines functionality, power, and flexibility. Python supports a wide range of applications.

**Pyhton** is an interpreted, interactive, object-oriented programming language. It combines functionality, power, and flexibility. Python supports a wide range of applications.

**Pyhton** is an interpreted, interactive, object-oriented programming language. It combines functionality, power, and flexibility. Python supports a wide range of applications.

**Pyhton** is an interpreted, interactive, object-oriented programming language. It combines functionality, power, and flexibility. Python supports a wide range of applications.

**Pyhton** is an interpreted, interactive, object-oriented programming language. It combines functionality, power, and flexibility. Python supports a wide range of applications.

**Pyhton** is an interpreted, interactive, object-oriented programming language. It combines functionality, power, and flexibility. Python supports a wide range of applications.

**Pyhton** is an interpreted, interactive, object-oriented programming language. It combines functionality, power, and flexibility. Python supports a wide range of applications.

**Pyhton** is an interpreted, interactive, object-oriented programming language. It combines functionality, power, and flexibility. Python supports a wide range of applications.

**Pyhton** is an interpreted, interactive, object-oriented programming language. It combines functionality, power, and flexibility. Python supports a wide range of applications.

**Pyhton** is an interpreted, interactive, object-oriented programming language. It combines functionality, power, and flexibility. Python supports a wide range of applications.
<table>
<thead>
<tr>
<th>Requirement</th>
<th>M</th>
<th>L</th>
<th>W</th>
<th>P</th>
<th>E</th>
<th>G</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simile</td>
<td>Framework for Integrated Water Management (but broader than water, so name will change soon)</td>
<td><a href="mailto:Tanya.Otjens@wur.nl">Tanya.Otjens@wur.nl</a></td>
<td><a href="mailto:Tanya.vanderwal@wur.nl">Tanya.vanderwal@wur.nl</a></td>
<td>Python</td>
<td>General Framework</td>
<td>CNC General Coupling Framework</td>
<td>Integrating Modelling Architecture</td>
</tr>
<tr>
<td>FIW</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODCOM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**SEAMLESS architecture**

- **M** - Multi-language component platform based on a declarative approach, specific to SEAMLESS.
- **L** - We will glue together different Python based frameworks and multi-language components (C, C++, Fortran, Java, R, Python, Matlab, ...) to obtain a flexible and scalable open-source architecture.
- **W** - Several of the requirements listed below pertain to functionality in layer 4. As may be clear from the above, we strongly believe that such functionality is best implemented on the foundation provided by layers 1 through 3.
- **P** - A large number of extension modules have been developed for Python. Some are part of the standard library of tools, usable in any Python program (e.g. the math library and regular expressions). Others are specific to a particular platform or environment (e.g. UNILP networking or X11) or provide application-specific functionality (e.g. image or sound processing).
- **E** - Python also provides facilities for introspection, so that e.g. a debugger or profiler for Python programs can be written in Python itself. There is also a general way to convert an object into a stream of bytes and back, which can be used to implement object persistence as well as various distributed object models.
- **G** - Our proposal for the SEAMLESS architecture is to develop a multi-language component platform based on a declarative approach, specific to SEAMLESS. We will glue together different Python based frameworks and multi-language components (C, C++, Fortran, Java, R, Python, Matlab, ...) to obtain a flexible and scalable open-source architecture.
- **Y** - On the technical side, many of our requirements are similar to yours. We require the declarative approach between modules; we have a need to integrate quantitative and qualitative information; we require application to both Unix and Windows environments; and we will need to accommodate both open and closed source modules within the system; and version control is of course a standard need.

### Supporting commitments in key areas such as data representation and visualization.

All fundamental data types, such as raster or time series, are defined within the data frames, which supports the definitions of new, compatible data types. The visualization framework allows the definition of "layers", each providing a visual representation of some type of data, such as rasters or polygons. Multiple layers can be placed on a single "view", such as overlaying a polygonal map on a raster. Views can be contained in "documents", such as arc and titles, each of which can be modified independently.

**TIME** includes a number of tools, which operate generally on objects, including an automated user interface generator and various model optimization tools. TME is developed on the Microsoft .NET platform and supports the development of models in a variety of languages, including Visual Basic .NET, Fortran 95.NET, C# and Visual J#. TIME is currently being used to develop a range of modelling applications, including a library of rainfall runoff models and a model supporting assessment of stream ecosystem health under various flow scenarios.