Enriching environmental software model interfaces through ontology-based tools

Ioannis N. Athanasiadis*
and Andrea-Emilio Rizzoli

Dalle Molle Institute for Artificial Intelligence (IDSIA),
USI-SUPSI,
Galleria 2, CH-6928 Manno,
Lugano, Switzerland
E-mail: ioannis@idsia.ch
E-mail: andrea@idsia.ch
*Corresponding author

Marcello Donatelli

Research Institute for Industrial Crop,
Italian Agriculture Research Council,
Via di Corticella 133,
40129 Bologna, Italy

and

IPSC MARS-AGRI4CAST,
Joint Research Centre – European Commission,
21027 Ispra (VA), Italy
E-mail: marcello.donatelli@jrc.ec.europa.eu

Laura Carlini

TSE S.r.l. Ingegneria e Impianti,
Via G. Galilei,
18 50021 Barberino Val d'Elsa (FI), Italy
E-mail: laura.carlini@icetindustrie.it

Abstract: Common practice has proven that software implementations of environmental models are seldom reused by broader communities or in different modelling frameworks. One of the reasons for this situation is the poor semantics of model interfaces. Model interfaces embrace a critical amount of the modellers' knowledge, but their software implementations can be considered as 'poor reflections' of modellers' perceptions, as they fail to represent the complexity of model assumptions in software terms. This paper addresses the problem by adopting an ontology-driven approach that aims to enrich software model interfaces with advanced semantics.

Keywords: knowledge-based software engineering; software component integration and reuse; declarative modelling; environmental informatics.

Reference to this paper should be made as follows: Athanasiadis, I.N., Rizzoli, A-E., Donatelli, M. and Carlini, L. (2011) 'Enriching environmental software model interfaces through ontology-based tools', *Int. J. Advanced Systemic Studies*, Vol. 4, Nos. 1/2, pp.94–105.

Biographical notes: Ioannis N. Athanasiadis holds a Diploma and a PhD in Electrical and Computer Engineering, both from the Aristotle University of Thessaloniki, Greece. Since 2005, he has been a Researcher with the Dalle Molle Institute for Artificial Intelligence in Lugano, Switzerland. His research interests include software engineering for ecoinformatics, ontologies and the semantic web, intelligent systems and software agents, agent-based social modelling and simulation, decision support systems, machine learning, and data and knowledge engineering.

Andrea-Emilio Rizzoli holds a Laurea in Electronics and Informatics Engineering (1989), and a PhD in Information and Automation Engineering (1993), both from Politecnico di Milano. He is a Senior Researcher at the Istituto Dalle Molle di Studi sull'Intelligenza Artificiale, in Lugano, Switzerland. His research interests include industrial and logistic chain system modelling and simulation, integration and reuse of simulation models, and decision support systems for natural resources management. Since 2005 he has also been a Professor with the University of Applied Sciences of Southern Switzerland (SUPSI).

Marcello Donatelli is a Senior Researcher at the Research Institute for Industrial Crops of the Italian Agriculture Research Council. His activity over the last 15 years has been focused on modelling in agrometerology and cropping systems. Over the last five years he has worked extensively on the development of software components for biophysical systems modelling, developing software designs to maximise transparency, reusability and extensibility. He is currently with the European Joint Research of the European Commission, leading the development of a modular platform for simulation via biophysical models of scenarios related to climate change and agriculture.

Laura Carlini was an Associated Researcher at the Research Institute for Industrial Crop of the Italian Agriculture Research Council. She has been working on model development and software prototyping in the area of agrometeorology, agro-ecology and agricultural production. She currently works in the area of renewable resources, focused on designing solar energy plants.

1 Introduction

Writing a model of an environmental system is a complex process, which aims at providing an abstraction of real world processes, using a given formalism, and exploiting a wide collection of techniques originating from general systems theory to economics and social sciences.

A model, being an abstraction, in order to tame the complexity of the real world, approaches its subject from a specific point of view; particular assumptions and hypotheses about the phenomena involved are made. We, therefore, neglect the full extent of causal chains and driving forces of the phenomena of interest and strive for simplification, focalisation and modularisation of the model construction process.

When we implement the model on a computer, we introduce more assumptions; more limitations (for instance, the model is forced to a discretisation) and, therefore, the software implementation of a model should be considered as a poor realisation of the original formalisation. Such an approximation states only implicitly the assumptions made for building it. For instance, the spatial discretisation of a model variable can only be inferred by a close inspection of the data type used to implement it.

During the last decades, a number of models have been designed and implemented, and it has become natural to assemble them together in order to try to address more and more complex problems. Integrated assessments are becoming increasingly common in environmental management and therefore we are faced with the problem of integrating models across scales and disciplines. This is neither an easy, nor a straightforward process.

Software Engineering promotes the concepts of reusing "components-off-the-shelf" (Szyperski et al., 2002; Egyed et al., 2005), distributed computing (Attiya and Welch, 2004), agent-based computing (Luck et al., 2005), service-oriented architectures and web services (Erl, 2004) to support the development of modular applications. The very same concepts are meant to be used to develop modular and integrated environmental software applications.

However, software integration is not the sole necessary condition for a proper assemblage of environmental models. In other words, if a set of (good) software model implementations is working together, this is not at all a sign that the compound model makes any sense from a modelling point of view and generates credible results. Different authors have tried to target the issue of quality assurance in the development of environmental models (Refsgaard et al., 2006; Jakeman et al., 2006), but their main focus is on the quality of the modelling process.

In this work we argue that sound integration of environmental models also requires automated coupling of the knowledge hidden behind each software implementation. In particular, in Section 2 we investigate a model structure and identify its knowledge elements, typically implicit both in the model interface and implementation. Section 3 focuses in the utilisation of ontologies for specifying model interfaces, while in Section 4 we present a demo web-based tool for communal ontology authoring for defining model interfaces. Section 5 presents a second tool for generating the source code of model interface data types, specified using ontologies, and the paper concludes with a discussion in Section 6.

2 Model knowledge and linking

2.1 The knowledge encapsulated in a model

The result of the modelling process is a formalisation that encapsulates knowledge related to both the interactions of the modelled system with its surrounding environment (model interface and data exchange), and the internal behaviour of the system (model equations, or endogenous variables). Consequently, a software component implementing a model will consist of two parts, the interface and the implementation. The interface defines the inputs, outputs and parameters of a model, while the implementation defines the model equations.

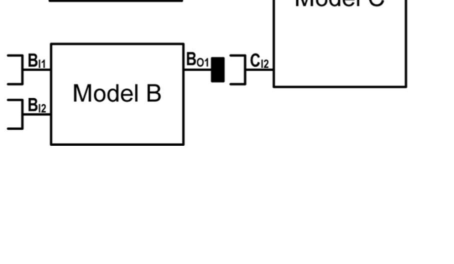
Declarative modelling aims to separate the algorithms, which execute the numerical solution of the model equations from the ‘declarations’ of the equations themselves, and the variables and parameters occurring in the equations. Prior work has focused on the equation part (Muetzelfeldt, 2004), whereas in this paper we concentrate on a declarative approach for describing a model interface to facilitate model linking and integration using ontologies. Ontologies provide a formal support to express conceptualisations (Gruber, 1993), and a number of tools support the creation of ontologies. Furthermore, model knowledge stored in the ontology can be used both for formal documentation and provide functionalities which go beyond the computation of model variables.

2.2 Sound model linking and integration

Easy model linking and integration is a key feature that is advertised by most modelling frameworks. However, we advocate that simple integration in software terms is not enough for sound model integration. A software implementation of an environmental model does not take into account the full semantics of the model interface. The model’s assumptions are not captured in a components’ software interface. The information associated with the inputs, states, outputs and parameters is limited to their data type. For instance, a typical software implementation expounds as model interface arrays of doubles, integers, and strings, whose context is described in the software documentation, or, even worse, only in the variable names. However, this practice requires that someone has to read the documentation in order to understand how to reuse this model properly. This is because the model’s knowledge related to its interface is not encapsulated in the actual interface of the model implementation in a self-explained fashion.

Consider, for example, the case depicted in Figure 1, where Models A and B are linked to another Model C. Model C is exposed to inputs C_{I1} and C_{I2}, which are to be linked to model outputs A_{O1} and B_{O1}. Let us assume, without loss of generality, that all these variables are simple floats. In software terms, integration can be achieved simply if both C_{I1} and C_{I2} are linked to any software component output that provides a float. However, from a modelling point of view, each model input or output is not simply a float; instead, it measures a specific quantity in a specific temporal and spatial context (i.e., it could be a car’s velocity or an ambient air pollutant’s concentration at ground level, and so on). Moreover, even if two models correctly link a variable expressing the same element, the model receiving the variable as an input may be able to handle only a sub-range of the values provided as outputs (due to model assumptions). It becomes evident that standard software interface conventions are not enough for encapsulating the full knowledge of the model interface.

Figure 1 A model linking example



The vision of reusing model software implementations as off-the-shelf components requires the assumptions on the model interface to be represented in the implementation in a machine-readable format. Following the previous example, suppose Models *A*, *B*, and *C* are supplied by diverse vendors. In order to achieve sound model integration, each linkage should be verified not only at the low level of data type matching (which is the unique requirement for software integration), but also against the actual semantics (context and assumptions) related to model interface. To elaborate a bit more, let Model *A* (of the previous example) expose a single float A_{O1} that represents the calculated rainfall output, while Model *C* has a water pressure input C_{I1}, also a float. Suppose that someone tries to make a link from A_{O1} to C_{I1}. In such a case, as both variables are represented as single floats, the integration is feasible in software terms, though it makes no sense from a modelling perspective. The same holds for less semantically diverse cases, where we could have model variables expressing the same concept, but with mismatches in characteristic times, units, pre- and post- conditions, temporal or spatial dimensions and sampling rates.

This discussion leads us to the conclusion that we need to express all the knowledge related to the model interface in a declarative way. This could be done by using an ontology, as we demonstrate in the following section.

2.3 Principles and methods

In this paper, we consider biophysical agricultural models in the context of the development of the Agricultural Production and Externalities Simulator (APES – http://www.apesimulator.it). APES is a modular simulation system, adopting a component-based architecture. A community of scientists from different disciplines contributed with model implementations that followed common design patterns. More than 20 scientists were involved in APES development, making scientific collaboration and knowledge sharing an important issue: Modellers needed to agree on common definitions of the entities exchanged by their models interfaces. In order to achieve a consensus, we followed a community-based, mediated approach, which involved ontological definitions, collaboration through scientific workshops and over the web, and tools for software development.

The first step was to identify key modelling assumptions, involving model types, units and dimensions, and translate them into an ontology specifying model interfaces. Then the community of developers continued the work, to populate the ontology with axioms specific to the model under development. This involved mediated workshops for exemplifying tools and goals. Then, after a couple of iterations and reviews, a consensus has been reached, with an ontology that specified model interfaces. The final step is to further exploit the definitions written in the ontology by exporting them to code.

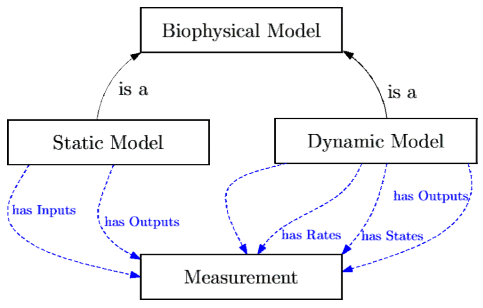
3 Towards an ontology for specifying model interfaces

3.1 Models and model types

In order to enrich model interfaces with advanced semantics, we developed an ontology, called the *Model Interface Ontology,* that aims to encapsulate our knowledge on the model interface in a declarative fashion. While we focused on agricultural biophysical processes, which occur through time and space, and usually are modelled using stocks and flows following the system dynamics approach, the *Model Interface Ontology* is generic and can be reused for other domains with similar properties. A model interface exposes both *stocks* (states) and *flows* (rates of inputs and outputs) and it can be accessed by a simulation engine (numerical integrator) for calculating the stocks as an accumulation of flows over the simulation time horizon.

These concepts are declared in the *Model Interface Ontology* as follows: we identify two types of models: *Static* and *Dynamic* models. The first kind of model does not expose any states and rates, as it is not required to be integrated over time. The opposite holds for the dynamic models. All inputs, outputs, states and rates of models are types of an abstract *Measurement* concept, which is used for defining their semantics in different contexts (space, time units, and so on). The Measurement class is detailed below. Figure 2, illustrates the relations between the two model types in the ontology.

Figure 2 The relations between the model type concepts of the model interface ontology (see online version for colours)



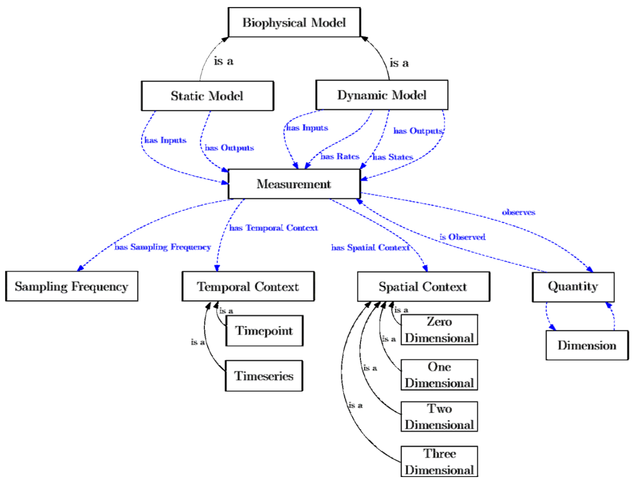
3.2 *Model interface elements as measurements*

The Measurement class is the key instrument for conceptualising the model interface elements. The Measurement class specifies the following properties of a model interface element:

- The observed quantity
- The spatial observation context
- The temporal observation context
- The sampling frequency
- Value conditions (minimum, maximum and default value and default unit).

A *Quantity* can be considered as the result of applying a physical dimension on a subject of interest. For example, *AirTemperature* can be considered as a physical quantity that represents the *Temperature* dimension of air. *Spatial* and *Temporal contexts* are used to define the dimensionality of a measurement in space and time. *Sampling frequency* associates the tempo-spatial dimension of a measurement with a sampling rate and grid size. Finally, *value conditions* are used for defining boundary conditions for a measurement's allowed values. An abstract view on the Measurement class and its relationships with the rest concepts in the ontology is presented in Figure 3.

Figure 3 The relations of the measurement concept (see online version for colours)



3.3 *Proof-of-concept and implementation*

The conceptual schema of the *Model Interface Ontology* can be used for specifying model interface elements as follows: consider a measurement called “*HourlyAirTemperature*”. It can be defined by referring to *AirTemperature* quantity, and be measured at a *point* in *space* and *time*, on an *hourly* basis, having as default unit degrees *Celsius* and be consistent to some value conditions (min, max, and default values). Consequently, such an instance of *Measurement class* can be attached to a model interface.

Note that the developed *Model Interface Ontology* has been realised using the Web Ontology Language (OWL, McGuinness and van Harmelen, 2004), through the Protégé ontology editor (http://protege.stanford.edu/plugins/owl/). OWL–DL expressivity was enough for conceptualising this domain. The specifications of units and dimensions were based on the SWEET ontologies (2006).

In the previous sections, we advocated the potential of publishing model interfaces in a declarative format and proposed an ontology for capturing the semantics of model interface elements. This approach was undertaken by the Seamless-IP project and the community of Agricultural Production Externalities Simulator (APES) modellers. A set of tools have been developed to enable modellers to:

- share their knowledge related to environmental model components and their interface variables
- exploit the knowledge stored in the ontology by generating source code in an automated fashion.

4 AgrOntologies: A web-based tool for collaborative ontology authoring

The process of setting up an ontology, and populating it with modellers’ knowledge was not straightforward. The major problems experienced, were related to managing modeller’s conflicting views and the complexity of the domain at hand. In order to tackle such issues and to facilitate knowledge elicitation within a community of more than ten modelling teams involved in APES, we built a web-based tool, called AgrOntologies, for communal ontology authoring.

The key requirements were to enable scientists to share their modelling knowledge at design time, enable collaboration over the web, and hide the complexity of an axiom-based system as OWL-DL. In order to achieve this we designed it as a web portal where scientists can register their model interfaces, review the definitions of other scientists and reuse the community the knowledge on units, dimensions and model interface elements. The implementation was based on Powl (Auer, 2005), a web based platform for collaborative semantic web development, which has been tailored for the needs of the *Model Interface Ontology* and the APES community.

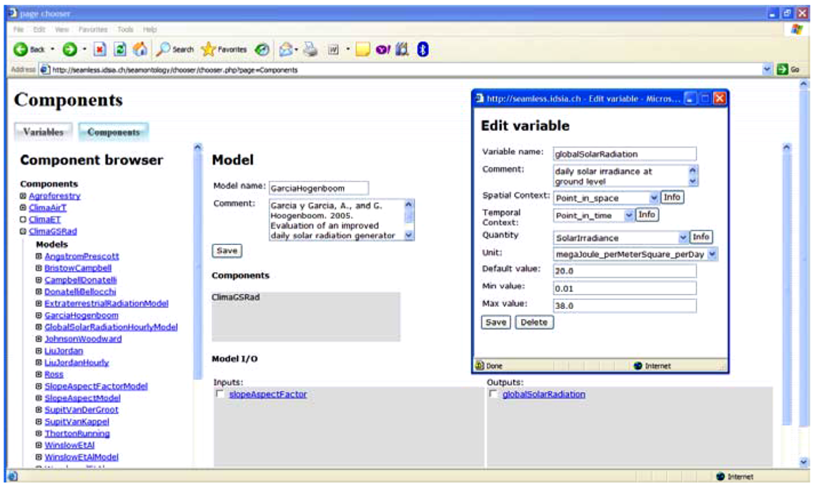
A key issue in this process is that modellers are required to make their model interfaces explicit and communicate them in a formal, yet comprehendible way to others. Through the AgrOntologies portal, a modeller can

- specify model variables in detail, or even reuse existing variables defined by others
- define model interfaces and ultimately
- put together models together in components.

Note that the AgrOntologies portal presents information to the users in a 'natural' way for them, not as they are represented within the ontology using description logics. In this sense, modellers are not required to be exposed to all the complexity of the internal ontology structure; rather, they are allowed to register their models through an easy to use portal.

A screenshot of the developed portal is shown in Figure 4: on the left is the component browser, which lists all models included in each bundle, while on the right we see the model specification that contains its name, comments, the components which is shipped with and model inputs and outputs. The pop-up screen on the top-right is the 'edit variable' panel through which the end user may define model variables as measurements (i.e., by setting spatio-temporal references, units and quantity).

Figure 4 The AgrOntologies portal screenshot (see online version for colours)


page chooser
File Edit View Favorites Tools Help
Back Search Favorites
Address http://seamless.idsia.ch/seamontology/chooser/chooser.php?page=Components
Go

Components

Variables | Components

Component browser

Components
- Agroforestry
- ClimaAirT
- ClimaET
- ClimaGSRad
  - Models
    - AngstromPrescott
    - BristowCampbell
    - CampbellDonatelli
    - DonatelliBellocchi
    - ExtraterrestrialRadiationModel
    - GarciaHogenboom
    - GlobalSolarRadiationHourlyModel
    - JohnsonWoodward
    - LiuJordan
    - LiuJordanHourly
    - Ross
    - SlopeAspectFactorModel
    - SlopeAspectModel
    - SupitVanDerGroot
    - SupitVanKappel
    - ThortonRunning
    - WinslowEtAl
    - WinslowEtAlModel

Model

Model name:	GarciaHogenboom
Comment:	Garcia y Garcia, A., and G. Hoogenboom. 2005. Evaluation of an improved daily solar radiation generator

Save

Components

ClimaGSRad

Model I/O

Inputs:	Outputs:
slopeAspectFactor	globalSolarRadiation

Edit variable

Variable name:	globalSolarRadiation
Comment:	daily solar irradiance at ground level
Spatial Context:	Point_in_space
Temporal Context:	Point_in_time
Quantity	SolarIrradiance
Unit:	megaJoule_perMeterSquare_perDay
Default value:	20.0
Min value:	0.01
Max value:	38.0

5 DCC: A tool for generating model source code

The use of the ontology that we collaboratively authored and reviewed with AgrOntologies, as a set of definitions (expressed as concepts and instances in the ontology) goes beyond the purposes of documentation, consensus building and model component linking. The attributes values associated with each variable can, in fact, be used to provide to the component information needed to test the adequacy of values at run time. This can be done via the implementation of the design-by-contract approach to test pre-conditions (e.g., Donatelli et al., 2006a, 2006b). Making available variables attributes in an implementation of model components has multiple uses, as it allows:

- validating inputs to the component
- using bounds for model parameters in automatic calibration
- defining sub-ranges of allowed variables to account for specific model limitation
- providing attribute values as simulation output for auto-documentation of results.

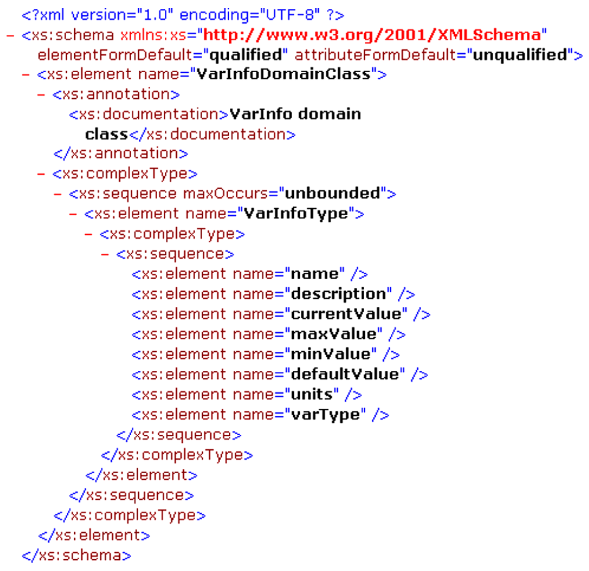
A software design which allows implementing the information available in components makes use of an abstract data type called the domain class, following the approach by Rizzoli et al. (1998). The domain class is characterised by a set of data attributes, which are the inputs, states, outputs of the model and a set of access methods to set and get the attribute values. The data attributes contain the numerical value, the variable's range, the default value, and the measurement units. Defining a domain class also allows setting the boundaries of the domain to be modelled, providing the information to model according to the approach chosen. Multiple models implemented in a component can make use of the same domain class.

The application Domain Class Coder (DCC) is a windows application which, from an input file extracted from the ontology application described in Section 4, generates the C# code of twin classes. Such classes are of the type to hold values, and a companion class to hold variables attributes. The former is an abstract class to be used as type in the component interface, which then allows extensions via sub-classing of its default implementation. The other class, conventionally referred to with the postfix VarInfo to the value class name, contains attribute values which are declared as static properties and have only the get access method. VarInfo values are used by a component to test pre and post conditions which uses the VarInfo type (CRA.core.preconditions.dll, available as the DCC, at http://www.isci.it/tools; DCC is available at the page XP Utils). The XML schema of the latter type is shown in Figure 5. From the XML schema it becomes evident that the information realised in the domain class is less compared to that stored in the ontology, but it is functional for the purpose described above.

Once the input file is loaded (either as an XML or as a tab separated ASCII file), the user can change minimum and maximum values to account for specific model limitations (if any) with respect to the values stored in the ontology. The user must also specify the domain class name, and the namespace of the class. The output is given by the C# code of the two classes described above, which implement interfaces which allow discovering types and attributes via reflection. The package (which can be downloaded) also contains a sample input file which allows generating the relevant classes.

When these classes are included in a component assembly, its content can be browsed via reflection using the application Model Component Explorer. This component allows discovering the domain classes, their attributes and types, and the VarInfo values for each attribute. The component is available in the same page of the DCC.

Figure 5 The XML schema of the VarInfo domain class (see online version for colours)



```
  <?xml version="1.0" encoding="UTF-8" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
  - <xs:element name="VarInfoDomainClass">
    - <xs:annotation>
        <xs:documentation>VarInfo domain
          class</xs:documentation>
      </xs:annotation>
    - <xs:complexType>
      - <xs:sequence maxOccurs="unbounded">
        - <xs:element name="VarInfoType">
          - <xs:complexType>
            - <xs:sequence>
                <xs:element name="name" />
                <xs:element name="description" />
                <xs:element name="currentValue" />
                <xs:element name="maxValue" />
                <xs:element name="minValue" />
                <xs:element name="defaultValue" />
                <xs:element name="units" />
                <xs:element name="varType" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

6 Discussion

Various Environmental Management Information Systems have exploited ontologies, mainly for information processing. Most of them focus on seamless integration of environmental data repositories, e.g., related to coastal zone management (Christophides et al., 1999), weather (Dance and Gorman, 2002), or water management (Felluga et al., 2003). More generic approaches for environmental data fusion, such as Infosleuth (Nodine et al., 2000), Buster (Neumann et al., 2001) and AISLE (Athanasiadis, 2006) utilised ontologies too.

However, none of the systems developed so far used ontologies for environmental model linking and model component integration. This is the major contribution of this paper, where we introduced ontologies as a medium for efficient model integration. The *Model Interface Ontology* was proposed for enriching environmental model interfaces in a declarative fashion. The MIO ontology consists of a first approach for specifying complex environmental model variables using rich semantics. Spatio-temporal aspects, units, quantities and dimensions are captured efficiently. Also, a clear path for building reusable software components was defined, and the use of ontologies, accompanied by a set of supporting tools, was exemplified. A clear pathway for developing software components and reusing them efficiently as components off-the-shelf was demonstrated, while taking into account the complexities and the richness of environmental models.

The principles and tools presented here have been successfully employed in the development of the APES system. The major obstacles we experienced reside with the human side of a complex modelling task. Engaging the community in such a unique exercise proved to be a hard task that required significant efforts for promoting the semantic modelling paradigm. However, with the tools developed we communicated the message that community-based ontology authoring can be as easy as navigating a portal, while the benefits of using to an ontology for defining model interfaces go beyond the system design and documentation.

Parallel efforts (i.e., Villa et al., 2006, 2009) are focusing on extending the current framework by specifying model equations using semantic modelling primitives. Ontology representations of both model interfaces and equations may lead us to a fully declarative modelling and simulation environment for ecological modelling.

Acknowledgements

The efforts leading to these results have received funding from the European Community's Sixth and Seventh Framework Programmes under grants "*Seamless IP*" (FP6-SUSTDEV-10036) and '*TaToo*' (FP7-IST-247893). The AgrOntologies portal tool was implemented by David Huber, AntOptima SA.

References

Athanasiadis, I.N. (2006) 'An intelligent service layer upgrades environmental information management', *IT Professional*, Vol. 8, No. 3, pp.34–39.

Auer, S. (2005) 'Powl – a web based platform for collaborative semantic web development', *Proc. of 1st Workshop Workshop Scripting for the Semantic Web (SFSW'05)*, Hersonisos, Greece.

Attiya, H. and Welch, J. (2004) *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*, 2nd ed., Willey, Hoboken, NJ, USA.

Christophides, V., Houstis, C., Lalis, S. and Tsalapata, H. (1999) 'Ontology-driven integration of scientific repositories', *Proc. of the Fourth Workshop on Next Generation Information Technologies*, Zikhron-Yaakov, Israel.

Dance, S. and Gorman, M. (2002) 'Intelligent agents in the Australian bureau of meteorology', *Proc. of the 1st International Workshop on Challenges in Open Agent Systems held at AAMAS'02*, Bologna, Italy.

Donatelli, M., Bellocchi, G. and Carlini, L. (2006a) 'Sharing knowledge via software components: models on reference evapotranspiration', *European Journal of Agronomy*, Vol. 24, No. 2, pp.186–192.

Donatelli, M., Bellocchi, G. and Carlini, L. (2006b) 'A software component for estimating solar radiation', *Environmental Modelling and Software*, Vol. 21, No. 3, pp.411–416.

Egyed, A., Müller, H.A. and Perry, D.E. (2005) 'Integrating COTS into the development process', *IEEE Software*, Vol. 22, No. 4, pp.16–18.

Erl, T. (2004) *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*, Prentice-Hall, Upper Saddle River, NJ, USA.

Felluga, B., Gauthier, T., Genesh, A., Haastrup, P., Neophytou, C., Poslad, S., Preux, D., Plini, P., Santouridis, I., Stjernholm, M. and Würtz, J. (2003) 'Environmental data exchange for inland waters using independent software agents', *Report 20549 EN. Institute for Environment and Sustainability*, European Joint Research Centre, Ispra, Italy.

Gruber, T.R. (1993) 'A translation approach to portable ontologies', *Knowledge Acquisition*, Vol. 5, No. 2, pp.199–220.

Jakeman, A.J., Letcher, R.A. and Norton, J.P. (2006) 'Ten iterative steps in development and evaluation of environmental models', *Environmental Modelling & Software*, Vol. 21, No. 5, pp.602–614.

Luck, M., McBurney, P., Shehory, O. and Willmot, S. (Eds.) (2005) *Agent Technology: Computing as Interaction*, AgentLink, Southampton, UK.

McGuinness, D.L and van Harmelen, F. (Eds.) (2004) *OWL Web Ontology Language Overview*, W3C Recommendation, www.w3.org/TR/owl-features/

Muetzelfeldt, R.I. (2004) *Declarative Modelling in Ecological and Environmental Research. European Commission Directorate-General for Research*, Position Paper no. EUR 20918. European Commission, Brussels, Belgium.

Neumann, H., Schuster, G., Stuckenschmidt, H., Visser, U. and Vögele, T. (2001) 'Intelligent brokering of environmental information with the buster system', in Hilty, L.M. and Gilgen, P.W. (Eds.): *Intl. Symposium Informatics for Environmental Protection*, Zurich, Switzerland, pp.505–512.

Nodine, M., Fowler, J., Ksiezyk, T., Perry, B., Taylor, M. and Unruh, A. (2000) 'Active information gathering in infosleuth', *International Journal of Cooperative Information Systems*, Vol. 9, Nos. 1–2, pp.3–28.

Rizzoli, A.E., Davis, J.R. and Abel, D.J. (1998) 'A model management system for model integration and re-use', *Decision Support Systems*, Vol. 4, No. 2, pp.127–144.

Refsgaard, J.C., van der Sluijs, J.P., Brown, J. and van der Keur, P. (2006) 'A framework for dealing with uncertainty due to model structure error', *Advances in Water Resources*, Vol. 29, No. 11, pp.1586–1597.

SWEET Ontologies (2006) *Semantic Web for Earth and Environmental Terminology (SWEET)*, http://sweet.jpl.nasa.gov.

Szyperski, C., Gruntz, D. and Murer, S. (2002) *Component Software – Beyond Object-Oriented Programming*, 2nd ed., ACM Press, New York, NY.

Villa, F., Donatelli, M., Rizzoli, A., Krause, P. and van Ewert, F.K. (2006) 'Declarative modelling for architecture independence and data/model integration: A case study', *In 3rd Biennial meeting of the International Environmental Modelling and Software Society*, Burlington, VT, USA, 9–12 July.

Villa, F., Athanasiadis, I.N. and Rizzoli, A.E. (2009) 'Modelling with knowledge: a review of emerging semantic approaches to environmental modelling', *Environmental Modelling and Software*, Vol. 24, pp.577–587.