International Environmental Modelling and Software Society (iEMSs)
7th Intl. Congress on Env. Modelling and Software, San Diego, CA, USA,
Daniel P. Ames, Nigel W.T. Quinn and Andrea E. Rizzoli (Eds.)
http://www.iemss.org/society/index.php/iemss-2014-proceedings

An architecture for integration of multidisciplinary models

Getachew F. Belete [a], **Alexey Voinov** [b], **Niels Holst** [c]

[a, b] University of Twente, Department of Geo-Information Processing, Netherlands,
[a] getfeleke@gmail.com, [b] aavoinov@gmail.com

[c] Aarhus University, Department of Agroecology, Denmark, niels.holst@agrsci.dk

Abstract: Integrating multidisciplinary models requires linking models: that may operate at different temporal and spatial scales; developed using different methodologies, tools and techniques; different levels of complexity; calibrated for different ranges of inputs and outputs, etc. On the other hand, integration of models requires us to address technical, semantic, and dataset aspects of interoperability. So we need a genuine techniques that enable us to integrate various domain specific models for interdisciplinary study. In this research work, we investigated best practices of System Integration, Enterprise Application Integration, and Integration Design Patterns. We developed an architecture of a multidisciplinary model integration framework that brings these three aspects of integration together. Service-oriented-based platform independent architecture that enables to establish loosely coupled dependency among various models is presented.

***Keywords**:* integration; architecture; SOA; loose coupling; framework; semantics.

1. INTRODUCTION

Model integration means linking models together into an operational model chain (Knapen *et al.* 2013), despite their differences in assumptions and semantics: the models may operate at different temporal and spatial scales, may have been developed using different methodologies, may have different levels of complexity, may have been calibrated for different ranges of inputs and outputs, etc. Therefore model integration requires different levels of interoperability: technically, the models should be able to 'talk to each other'; semantically, differences in conceptual representation among models need to be mediated; and at dataset level, the output dataset of one model should be converted to match what is used as input by the next model in the chain.

Since we are integrating a wide range of independently built models we need to establish "few well-known dependencies"(Rosen et al., 2008) among them as a basis for loose coupling. Loose coupling enables late binding, a situation in which components can be swapped with other components (Seemann 2012). In this research work we would like to establish loose coupling of models using SOA, since in SOA-based loose coupling, services "impose low consumer coupling requirements and are themselves decoupled from their surrounding environment" (Erl *et al.* 2009). SOA is a design pattern "which is composed of loosely coupled, discoverable, reusable, inter-operable platform agnostic services in which each of these services follow a well-defined standard. Each of these services can be bound or unbound at any time and as needed" (Jamil 2009). Which implies, SOA is an appropriate means to implement multidisciplinary model integration to the degree, that it even

dissolves the problem: “integration as a concept begins to fade within service-oriented environments” (Erl 2008b).

SOA-based integration can be implemented using either service-based components or web services. Web service exposes public capabilities as operations “without any ties to proprietary communications framework” (Erl et al. 2009). However, “components typically rely on platform-specific development and runtime technologies” (Erl 2008a). Web service-based SOA has the benefit that it can resolve possible problems in message communication by using the 'reliable-messaging feature' of web services (Bilorusets et al. 2004). Besides, SAWSDL (Semantic Annotations for WSDL and XML schema) enables us to annotate WSDL descriptions of web services with pointers to semantic concepts (Kopecky et al. 2007), i.e. operations, parameters, and data types can be annotated using ontologies (Paulheim 2011). Such feature of incorporating contextual and meta-model information of models gives SOA based integration framework the potential to link interdisciplinary knowledge.

2. RELATED WORKS

The SEAMLESS Integrated Framework was developed for an ex-ante, integrated assessment of agro-environmental policies and agro-technological innovations in the European Union (EU). A Graphical User Interface based software which reflects the joint knowledge of participating models, was developed for the technical integration of models. Models were componentized using OpenMI component standard(Moore & Tindall 2005). A framework that enables the execution of componentized models in model chains was also provided (Janssen 2009). The Community Surface Dynamics Modeling System (CSDMS) is another component-based tool for model integration. In CSDMS, stand-alone models were made plug-and-play components by implementing the Common Component Architecture (CCA) standard (Armstrong *et al.* 1999). CSDMS provides a suite of tools developed for the technical integration of plug-and-play models following this standard (Peckham *et al.* 2013). Language interoperability among CSDMS componentized models is provided through a language interoperability tool known as Babel (Babel 2012).

On the other hand, Goodall *et al.* (2013) had used web services to couple climate and hydrological models which were deployed on different computing platforms. While it is a good progress, the usage of SOA benefits was limited to only creating interoperability. Generally, what makes this integration framework different from other related works is being SOA based framework:

- “processes and applications are no longer coded as complex program structures, but are orchestrated as independent, distributed service calls”(Schmutz *et al.* 2010),
- no additional language interoperability tool is required since “vendor diversification is especially attainable through the use of web services”(Erl *et al.* 2009),
- loose coupling is provided through message based communication - which avoids the need for persistent connections and minimizes coupling requirements,
- semantic mediation and dataset conversion tasks are facilitated by the message based communication since “message contents can be supplemented with activity specific metadata that can be interpreted and processed separately at runtime.”(Erl et al. 2009).

3. METHODOLOGICAL APPROACH

The methodology we used to develop the architecture of the integration framework was to investigate best practices of System Integration, Enterprise Application Integration, and Design Patterns in general; and SOA Principles and SOA Design Patterns in particular. Design patterns and best practices served us a means to solve integration challenges based on past experience and on

available alternative solution templates. So we identified techniques, that enable models to express the input, output, and functionalities they provide, and that limit the dependencies among the models. Methods to assemble and orchestrate various modules from different models, and to accommodate semantic and dataset mediation with accompanying utilities. As a result we have come up with a model integration architecture, that can accommodate a wide range of multidisciplinary models.

4. ARCHITECTURE OF THE MODEL INTEGRATION FRAMEWORK

Several design patterns already exist for system integration but model integration requires special attention concerning the technical, semantic, and dataset aspects of integration. In this integration framework we have tried to intermingle the three aspects of model integration with appropriate, SOA-based system integration design patterns. The integration architecture (shown in Fig 1) aims at integrating interdisciplinary, qualitative and quantitative models which can be at various spatial and temporal scale without the need for rewriting underlying models. It consists of resource layer, legacy wrapper layer, wrapper web service layer, mediation layer with its accompanying resources, and user interface layer. The model integration framework enables model integration through file transfer, shared databases, and message exchange in a loosely coupled way. The integration architecture can evolve in accordance with future changes of models; it is a vender neutral architecture which can be implemented with various options, and being a composition centric architecture, it enables the agile assembly of services.

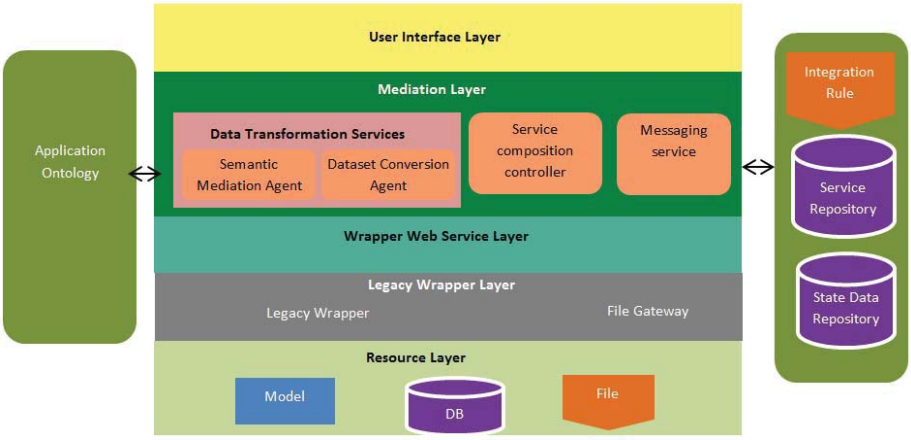

Figure 1: Conceptual architecture of the model integration framework

4.1. Resource Layer

The resource layer holds models, databases, and files which need to be integrated to assess interdisciplinary studies like low carbon economy scenarios. Each of them can be developed using different tools and techniques, and can be located anywhere in any appropriate hardware and software computing platforms.

4.2. Legacy Wrapper Layer

Wrappers, that provide a new calling interface to existing code, can assure language interoperability and can convert existing models into interoperable components (Peckham *et al.* 2013). The same applies to participating databases and files. Every participating model and database is wrapped by non-standardized legacy wrappers and files by using file gateways. These non-standardized legacy wrappers and file gateways will serve as foundation for the next level standardized wrappers. They also serve as a façade logic (Erl 2008a) to accommodate new versions of participating models by abstracting the changes.

4.3. Wrapper Web Service Layer

The aim of the web service wrapper layer is to further wrap the underlying non-standardized wrapper layer with a standardized service contract that extracts, encapsulates, and possibly eliminates legacy technical details. Basically standardized service wrappers can also be developed by using components which are developed as services. But when components are built as services (1) the service contract must be developed using the technologies used to build the component itself (in our case, with the same technology with the underlying legacy wrapper is developed); (2) the service can only be used by consumer programs that were developed using the same technology, or we have to use some bridging and transformation technology (Erl 2008a). However, the service contract should not be technology coupled, so that wrapper services can be developed using a mixture of different technologies, e.g. Java, .NET, etc. depending on the ease of implementation. Thus, to achieve platform neutrality we chose to use web services instead of components to develop wrapper services. Using SOA with web services as model wrappers furthermore enables us to utilize SOA abstraction design principles to hide technology, programmatic logic, and the underlying functions used to build the various models.

In service based system development, although building the service contract ahead of the implementation logic is the recommended approach, we cannot follow it since we are converting already built models into services. However, "by decoupling the service contract, the service implementation can be evolved without directly impacting service consumers. This can increase the amount of refactoring opportunities and the range of potential consumer programs (and corresponding reuse)" (Erl 2008a). The model integration framework is based on the principle of distributed computing, in which functionalities of different models are linked together to solve other 'bigger' problems (Fig 2). To realize this we have to maximize the composability capability of models by decomposing them into a number of low granularity services, each targeted to solve a clearly defined problem (the Single Responsibility Principle, Martin, 2006). The service decomposition depends on the underlying logic of the corresponding functionalities, and it helps us to have a number of clearly described services instead of one monolithic big service. There may not be planned composition for a service at hand but wrapper services need to be developed for each responsibilities. In designing services for better service composition, there are two kinds of couplings we must avoid to assure that the goals of SOA are fulfilled: (1) Direct consumer-to-implementation coupling must be avoided by contract centralization, so that service consumers will not access the wrapper components directly, (2) Indirect coupling of consumers to specific functionalities, technology, and implementation resources must be avoided by applying the service abstraction principle in service contracts.

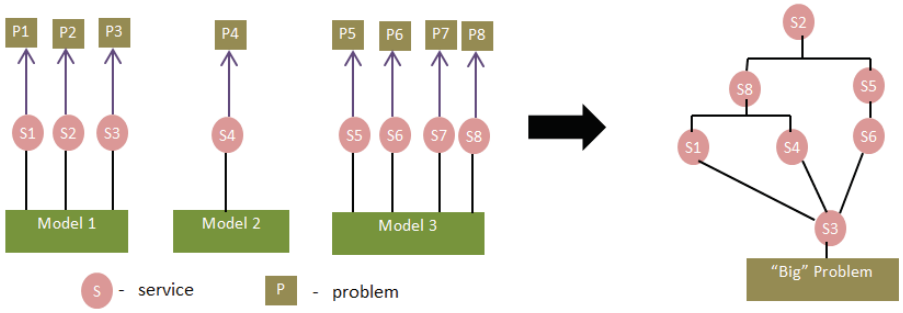

Figure 2: Granularity in service decomposition and service composition

4.4. Mediation Layer

The mediation layer consists of services that manage service composition, routing, reliable messaging, input-output mapping, and data transformation between service based models. It is a

middleware dedicated to provide interoperability. The services on the mediation layer should be implemented as agnostic and reusable services; to assure this they should be: generic in logic, and independent from technology and application platforms. Some of the mediation services (like semantic mediation and dataset integration services) shall be implemented as event-driven service agents (Erl 2008b) that respond automatically to predefined conditions without invocation via a published contract. This will help to decrease the number of service invocations in service composition and cause less performance strain.

4.4.1. Data transformation services

The data transformation task in this model integration framework is broadly divided into semantic mediation and dataset conversion. Although it is difficult to provide generic data transformation utility for all kinds of data transformation, we need to dedicate separate services for each kind of data transformation. These services need to be implemented as service agents with mediating features. Sometimes the data transformation task may also require action while having only incomplete knowledge, and with various levels of uncertainty. A data transformation service with artificial intelligence capability could possibly understand data transformation patterns and handle uncertain data.

a) **Semantic Mediation Agent**: The task of the semantic mediator is to bridge the semantic differences among models. How? From the meta-model information of the models, the semantic mediator can get the contexts of the sender and receiver models. Based on the available contextual information, the semantic mediator will amend a message sent by a model in a way which can be understood by the receiver model. In some of the cases, the semantic mediation may be variable mapping, i.e. when simple mapping of a variable is enough to create shared understanding between models. But in most of the cases we will use an ontology for semantic mediation; since (1) ontologies facilitate content-based access, communication and integration across different systems (Calı et al. 2005); (2) ontologies are based on first order logic upon which a computer can reason (Janssen 2009); and (3) ontologies enable the developer to practice a “higher” level of reuse, which is knowledge reuse (Wang et al. 2002). To realize this, a model interface ontology that describes the parts of the models, that will be involved in the integration process, needs to be developed. The semantic annotation of every member model can be made available using the corresponding SAWSDL since they “add hooks that lets WSDL components point to their semantics” (Kopecky et al. 2007).

b) **Dataset Conversion Agent**: By dataset integration of models, we let “data produced by one model be a meaningful input to another model, usually operating data at a different temporal and spatial scale” (Knapen et al. 2013). This may require processes, like estimation of missing input data by interpolation or extrapolation, aggregation, disaggregation, mapping, calibration, transformation, projection, categorization, and the facilitation of repeated runs of a model for different regions/time intervals (i.e. for a scaling process). The dataset conversion agent performs such conversions on data, based on the contextual information of the provider and consumer services, which is made available through service contracts and the service repository.

4.4.2. Service Composition Controller

After componentization, run-time composition is a major issue since it has significant effect on the scientific validity of the output of the integration. In this context service composition is the process of linking a number of services (models) to accomplish the aim of a given integration scenario. The service composition controller is a service which validates the workflow given by the user against the integration rule; and it invokes the corresponding series of services accordingly by avoiding point-to-point connections among them. It is responsible for managing the composition logic and orchestration.

In other words, it is a middleware platform that assures process centralization (Erl 2008a) using the integration rule. Various runtime challenging scenarios such as error and exception handling are also addressed by the composition controller.

4.4.3. Messaging Service

In the integration framework, in order to avoid persistent connections and to minimize the coupling requirement between services, all interaction takes place through messaging. A messaging service is part of the mediation layer and a service by itself, which is dedicated to reliable delivery of messages to the appropriate services, which could be located remotely.

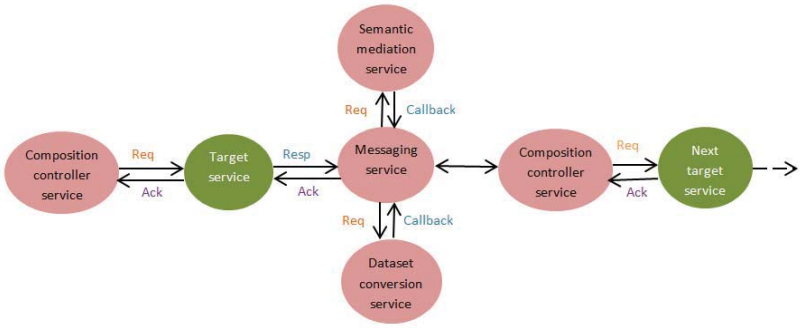

Figure 3: Messaging pattern of the proposed framework

As shown in Fig 3 the message exchange pattern between models (target services) and mediation services is a one-way pattern, often called “fire and forget” pattern (Josuttis 2007) with acknowledgement. The reason we selected this pattern is, that we are integrating independently built models, and the response does not necessarily need to go to the one who invokes it. The message exchange pattern between data transformation services and the messaging service is a request/callback message pattern, which is a non-blocking request/response type which enables the data transformation output be delivered asynchronously.

As a rule, service invocation (request) to models always originates from the UI layer and is realized by the service composition controller; correspondingly, models always deliver their outputs back to the messaging service. At design time, this rule enables that, whatever the number participating models in the framework, each wrapper web service will only need to implement only one connection and interface. During run-time it assures non-blocking communication between models and mediation services, which will prevent performance problems. Messaging will usually involve supplementing the message content with metadata, which will be used for semantic and dataset conversion. For reliable messaging the messaging service uses a state data repository to store relevant information.

4.4.4. Application Ontology

Since we are integrating multidisciplinary models, it is obvious that models can come from several domains. Therefore, developing ontologies for all relevant domains will be unrealistic. On the other hand, an ontology is crucial for automated semantic mediation. An application ontology, which “define concepts from a domain that are required for one application, as well as, optionally, some specific extensions that are needed for that application” (Paulheim 2011), can be implemented as part of the integration framework. The scope of application ontology will be delimited by concepts that are involved in message exchanges.

4.4.5. Integration Rule

The composition of services is guided by a designated service composition controller (4.4.2). The service composition controller uses an XML-based integration rule to validate user-proposed

compositions. The integration rule serves as an element for definition of workflow logic that dictates the flow of action and data in composing services. It enables the definition of inter-capability interactions between services at design time. The reason we need design-time specification of linking potential between service capabilities is logical and contextual. Furthermore, input-output data alignment of service capabilities should be identified, and the accompanying semantic and dataset mediation functionality should be implemented. Suppose we have a Rainfall Runoff model, which predicts the amount of runoff; and suppose there is a River model which takes the amount of runoff as input for its Lateral Flow attribute, i.e. as in Moore *et al.* (2005). Then, the service composition potential in the integration rule might look as shown in Fig 4.

```
<rule>
  <model name="Rainfall runoff">
       <service name="runoff">
          <capability name="runoff">
               <input_for>
                   <model name="River" service="river" capability_name="lateral inflow">
                   .....
               </input_for>
           </capability>
           .....
       </service>
  </model>
  .....
</rule>
```

Figure 4: Instance of integration rule

4.4.6. Service Repository

One of the objectives of this framework is to make multidisciplinary models available for researchers and scientists in a way, so that they can easily navigate through existing model inventory and perform linked model runs to assess different possible scenarios. To realize this objective, detail meta information of models can be made available through a centralized service repository for design-time discoverability. And the framework will have querying and filtering utility to discover the available service alternatives in the service repository.

4.4.7. State Data Repository

Regarding state data management, since we are integrating independently built models, we did not expect a high need of state management between participating models. However, state management will be needed in relation to mediation services and orchestrated task services. Due to these different cases and for better performance: (1) database–centric state deferral option for mediation services and orchestrated task services, and (2) message-based state deferral option for maintaining state data of models can be used (i.e. whenever needed). The message-based state deferral option can be managed as SOAP attachment.

4.5. User Interface Layer

The user interface layer is the part of the technical integration which interacts with users, and which shields the underlying technical, semantic and dataset conversion works. It enables users to select workflow, run simulation, and query data.

5. DISCUSSION AND CONCLUSION

Since the proposed framework is comprised of a number of constituents, we suggest the following points will have crucial effect in the implementation process and on the usability of the framework. The first major challenge to face in implementing this framework is getting detailed documentation and meta-model information about participating models. A model with: end user manual, technical documentation, well commented model source code, sample input-output data, and a cooperative

contact person is an ideal situation for implementing the integration. The less in availability those resources will be, the more challenging it will be to incorporate the model into the integration framework.

The other possible major challenges in implementing this architecture is in wrapping models with web services. For instance: in one scenario a model can be used for vertical integration with another model, i.e. model 1 finishes its execution and passes its output to model 2. In another scenario model 1 may be required to make horizontal integration with model 2 or model 3, i.e. it has to make two-way communication with the other model at a certain time steps. During wrapping we need to identify the possible 'points of integration' of models; the existing input-output points of the model may not be sufficient points for integration. Even the process can be more challenging when the model involves several interacting parties in it, e.g. the case of General Equilibrium Models. This may need understanding the whole implemented code, which may also require learning unfamiliar programming languages.

Handling human-ware of integration by designing user friendly and intuitive user interface is another critical challenge in implementation. It is obvious that a framework with minimal installation and configuration requirement, and with high accessibility and availability feature will have relatively higher usability. On the other hand, users of the integration framework can be scientists, researchers, stakeholders, policy makers, etc. A user who is expert in one domain can be layman in another domain. The technical complexity of the framework should not be deterrent for usage. In addition the ways, in which input is provided and output is presented in the framework, should be at least comparable with the user interface of the underlying models.

Application ontology development is the other area in realizing the framework. It is an iterative and incremental process which will be performed whenever new models join the integration framework. The process involves: identification of concepts with appropriate granularity, refinement and structuring of the identified concepts, identifying lexicon, organizing the terms according to conceptual hierarchies and structuring them with attributes and axioms, and then encode the ontology in a formal language. We can see that with the current trend of poor documentation of models, identifying concepts to be involved in the integration process may require huge effort and time. Besides, since application ontology is to be implemented, a concept which was left out in linking model 1 with model 2 could be found very important when model 3 joins the framework later on. Such issues should be resolved without disturbing the functioning of the existing semantic network. Another worry is that semantic mediation by ontology can have side effects on the performance of the system. To minimize its effect on performance, strategies like fetching 'relevant' semantic content during initialization together with caching (Weichselbraun *et al.* 2011) can be used. Generally to avoid the possible drawback in performance, appropriate data transfer techniques should be considered in serializing and transporting big data.

In converting the output of one model to make input for another model, there is associated uncertainty. If the data transformation needs only mapping out variables of the first model to input variables of the next model, we can be certain in data transformation process. But if besides mapping, the linking of models requires spatial or temporal up or down scaling, we will definitely have uncertainty. In some cases the level of uncertainty could possibly force us to decide not to incorporate the model into the integration framework. Identifying and handling uncertainty in data transformation process will have significant impact on the acceptance of the integration output. Quantifying the associated uncertainty can possibly contribute to the reliability of the framework.

Error and exception handling is the other area that requires huge effort in implementing the framework. Error and exception handling is not uniform and standardized among models. How do we handle error created by a model (inside a model) and error created by the framework? How do we report exception produced by the underlying models? Which kinds of errors are tolerable and which should result in interrupting model chain run? To answer these and other related questions we need to understand how error and exception is managed in each of participating models. We need also to separate mechanisms for handling error and exception produced by the framework and by the models.

Finally considering the above mentioned implementation related issues, we conclude that the use of SOA with web services can facilitate the model integration effort since the intrinsically interoperable

nature of web services enables the establishment of loose coupling among disparate multidisciplinary models. Besides, the discoverable nature of web services makes available the meta-model information of models for the underlying semantic and dataset mediation tasks. Although both the semantic and dataset mediation require additional research, the integration can only be realized when semantic mediation and dataset conversion are made available together with the technical integration work. In this research work we have shown how the three aspects of model integration can be smoothly synchronized using SOA based framework by applying appropriate integration design patterns and best practices.

Acknowledgments

G. F. Belete and A. Voinov was supported by COMPLEX – Knowledge Based Climate Mitigation Systems for a Low Carbon Economy Project, EU 7th Framework Program, Theme [env.2012.6.1-2], Grant agreement 308601.

References

Armstrong R, Gannon D, Geist A*, et al.* (1999) Toward a common component architecture for high-performance scientific computing, 115-124.

Babel (2012) *Babel: High-Performance Language Interoperability.* http://computation.llnl.gov/casc/components/#page=home

Bilorusets R, Box D, Cabrera LF*, et al.* (2004) Web services reliable messaging protocol (WS-ReliableMessaging). *BEA Systems, IBM Microsoft, Tibco*.

Calı A, Calvanese D, Grau BC*, et al.* (2005) State of the art survey. Technical Report WP1–Assessment of Fundamental Ontology Based Tasks, FP6-7603 Thinking ONtologiES (TONES) project.

Erl T (2008a) *SOA design patterns* Pearson Education.

Erl T (2008b) *Soa: principles of service design* Prentice Hall Upper Saddle River.

Erl T, Karmarkar A, Walmsley P*, et al.* (2009) *Web service contract design and versioning for SOA* Prentice Hall.

Goodall JL, Saint KD, Ercan MB*, et al.* (2013) Coupling climate and hydrological models: Interoperability through Web Services. *Environmental Modelling & Software* **46**, 250-259.

Jamil E (2009) What really is SOA. A comparison with Cloud Computing, Web 2.0, SaaS, WOA, Web Services, PaaS and others. *White Paper, Soalib Incorporated*.

Janssen SJ (2009) *Managing the Hydra in integration: developing an integrated assessment tool for agricultural systems* Wageningen Universiteit (Wageningen University).

Josuttis N (2007) *SOA in Practice* O'reilly.

Knapen R, Janssen S, Roosenschoon O*, et al.* (2013) Evaluating OpenMI as a model integration platform across disciplines. *Environmental Modelling & Software* **39**, 274-282.

Kopecky J, Vitvar T, Bournez C, Farrell J (2007) Sawsdl: Semantic annotations for wsdl and xml schema. *Internet Computing, IEEE* **11**, 60-67.

Martin M, Martin RC (2006) *Agile principles, patterns, and practices in C#* Pearson Education.

Moore RV, Tindall CI (2005) An overview of the open modelling interface and environment (the OpenMI). *Environmental Science & Policy* **8**, 279-286.

Paulheim H (2011) *Ontology-based application integration* Springer.

Peckham SD, Hutton EW, Norris B (2013) A component-based approach to integrated modeling in the geosciences: The design of CSDMS. *Computers & Geosciences* **53**, 3-12.

Schmutz G, Liebhart D, Welkenbach P (2010) *Service-oriented Architecture: An Integration Blueprint* Packt Publishing Ltd.

Seemann M (2012) *Dependency injection in. NET* Manning.

Wang X, Chan CW, Hamilton HJ (2002) Design of knowledge-based systems with the ontology-domain-system approach, 233-236.

Weichselbraun A, Wohlgenannt G, Scharl A (2011) Applying Optimal Stopping Theory to Improve the Performance of Ontology Refinement Methods, 1-10.