International Journal of Geo-Information

MDPI

Article

A Formal Framework for Integrated Environment Modeling Systems

Gaofeng Zhang [1,2], Yan Li [2], Chong Chen [2], Ri Zhou [2], Dan Chen [2] and Qingguo Zhou [2,*]

1 School of Physical Science and Technology, Lanzhou University, Lanzhou 730000, China; zhanggaof@lzu.edu.cn

2 School of Information Science and Engineering, Lanzhou University, Lanzhou 730000, China; ynali@lzu.edu.cn (Y.L.); chench2013@lzu.edu.cn (C.C.); zr@lzu.edu.cn (R.Z.); 7321582@163.com (D.C.)

* Correspondence: zhouqg@lzu.edu.cn; Tel.: +86-931-891-2025

Academic Editors: Jason C. Hung, Yu-Wei Chan, Neil Y. Yen, Qingguo Zhou and Wolfgang Kainz
Received: 30 October 2016; Accepted: 10 February 2017; Published: 17 February 2017

Abstract: Integrated Environment Modeling (IEM) has become more and more important for environmental studies and applications. IEM systems have also been extended from scientific studies to much wider practical application situations. The quality and improved efficiency of IEM systems have therefore become increasingly critical. Although many advanced and creative technologies have been adopted to improve the quality of IEM systems, there is scarcely any formal method for evaluating and improving them. This paper is devoted to proposing a formal method to improve the quality and the developing efficiency of IEM systems. Two primary contributions are made. Firstly, a formal framework for IEM is proposed. The framework not only reflects the static and dynamic features of IEM but also covers different views from variant roles throughout the IEM lifecycle. Secondly, the formal operational semantics corresponding to the former model of the IEM is derived in detail; it can be used as the basis for aiding automated integrated modeling and verifying the integrated model.

Keywords: Integrated Environment Modeling (IEM); formal method; operational semantics; unified view; Finite State Machine (FSM)

1. Introduction

Over the past 30 years, Integrated Environment Modeling (IEM) has become more and more important for environmental studies as it provides the ability to supply holistic views and solutions for environment science coupled with ecology, economy, and social activities [1–4]. Assisted by advanced computer science and technologies, many IEM systems have been developed to provide technical support for IEM.

IEM systems have undergone approximately three stages since the emergence of IEM. The earliest IEM systems, and even many new systems, are strictly integrated following to the classification proposed by Voinov and Shugart [5]. In such a system, the models, data, and analysis algorithms are tightly coupled together. Although most of these systems are effective and efficient, they are too tightly coupled to be reused for similar research, modeling, and analysis, which reduces our ability to efficiently develop new investigative paradigms. Therefore, the second stage focused on adopting modular technologies in system development. Modularized software models and analysis algorithms were developed, allowing for more loosely coupled systems. When new studies or applications are encountered, these modules can be abstracted from the original systems and integrated together into a new system with minimal adaptation. Thus, models and algorithms are more reusable and the development efficiency, as well as the quality, of the new system is promoted.

In the last 10–15 years, with the pressing need of environmental sustainable development, the IEM system has become the basis for applied uses as well as research [6]. This requires the IEM system to be developed more efficiently with much better quality; hence, IEM systems evolved into the third stage. Advanced technologies for software development have been adopted to support these requirements during this stage. Models and algorithms are developed as independent, self-contained software components, which are managed in a model or algorithm library. Each component offers one or more interfaces, through which different models and algorithms are woven together to simulate complex phenomena or processes. In this stage, the components of models and algorithms are developed independently. Achieving high reusability, loose coupling, and expandable structure of IEM systems is the main objective. At the same time, to make integration easier, more efficient, and more correct, some support tools have been specially developed. These tools aid IEM by helping build modular components and may be libraries or frameworks, such as MMS [7], OpenMI [8], OMS3 [9,10], ESMF [11], IWRMS [12], GMS/WMS/SMS BASINS [13], CSDMS [14], and SEAMLESS [15]. These tools try their best to relieve integrators from complicated software development techniques, allowing them to concentrate on the model's interaction and connection and reflecting the relationships of corresponding physical objects in the real world.

The IEM system has continually and promptly adopted many of the latest and advanced computing models, architectures, and techniques. Recently, Multi-Agent System (MAS) [16,17], grid computing [18,19], Service-Oriented Architecture (SOA) [20,21], and cloud computing [10,22] have also become involved. These new techniques make the models less dependent on each other. It has also become possible for many models to be integrated together no matter where they are deployed.

Nevertheless, the growth of the number of models in one system also increases its complexity. More models lead to more interaction. In addition to the dataflow, there is the control flow, which functions to smooth the gaps across models' semantics and implementation. Thus, the integrated model of IEM system development has a complex and even dynamic structure, which implies that the models and dataflow may themselves function as variables during the simulation. Consequently, it becomes difficult to precisely describe the structure of the integrated model.

Yet it is critical to correctly describe the structure of the model in order to reduce ambiguity, and ensure the model runs smoothly with its dynamic structure. We note that formal methods of developing computer systems use mathematically based techniques to describe system properties and can provide frameworks to specify, develop, and verify systems in a systematic manner [23]. Thus, research into defining formal developmental methods may help IEM as well; we focus on using some formal methods to enrich the IEM system in this paper.

Although there is little direct research on formal descriptions for developing IEMs, there have been many studies that enlighten our work. Argent [24] provided an overview of IEM that covered requirements, modeling, integration, development, frameworks, practice, and applications. Laniak et al. [6] summarized the landscape of IEM as containing four independent elements—applications, science, technology, and community—and proposed its roadmaps, which corresponded to each element. In addition, Argent et al. [25] also supplied an evaluation framework that includes conceptual ease, ease of development, support for model development, and run time features, etc. Voinov et al. [6,26] stressed the role of data in IEM and proposed that it is the data that link different models together and, most importantly, distinguish IEM from pure software composition. Rizzoli [27] focused on the semantics of the model interface link models together smoothly, while Schmitz et al. [28] focused in detail on cooperation between models with different temporal scales. Kragt et al. [29] emphasized the role of modelers and provided a framework through which environmental modelers can guide more successful integrative research programs. Lloyd et al. [30] used software engineering methods to illustrate that models with lower framework invasiveness tended to be smaller, less complex, and have less coupling. Our own practices of integrated modeling [31,32] also suggested that there should be some technologies to describe integration more precisely.

Inspired by these significant studies, this paper aims to construct a formal framework to improve the development efficiency of integrated model and IEM systems. Although there have been many formal methods designed to tackle the composition of components, services, and agents [33,34], there are few formal methods to describe the dynamics of the integrated models' structure at the level of application. Thus, this paper proposes actor-oriented-like semantics with which the static and dynamic attributes of models, rather than those of the software components, services, and agents corresponding to the model, are defined.

Two main IEM views, different but related, are proposed to satisfy different objectives. Graph-based semantics are used to supply the visual semantics for integration, from the perspective of the integrators, and runtime control, from the perspective of the IEM system. The interaction among the models is represented by an intuitional multidigraph with ports, in which the dataflow itself is represented with the edge, models with vectors, and variables with ports.

For conceptual ease of reuse, another, unified, view of the model, based on a set of integrated algebra, is defined. Reuse is one important factor of IEMs: integrated models, as well as related knowledge, should be easily reused as this is the goal of integration. To simplify reuse of the integrated model, this paper proposes a hierarchical FSM (HFSM)-based integrated algebra for modeling the control flow of IEM, such that a unified view of both the simple model and integrated model is achieved in our semantic framework.

There have been some studies to tackle the hierarchal structure formal analysis, such as analyzing MAS with a hierarchal Petri-net [35] and model checking of a hierarchical FSM [36]. This implies that the complexity that arises from the unified view of the model is affordable. Some analogous studies have been conducted for decades. In the DSP (Digital Signal Processing) area, dataflow (DF) is combined with HFSM to improve the quality (such as static schedule) of the system [37]. In some science workflow systems (e.g., Kepler) and simulation frameworks (e.g., Ptolemy, which is Kepler's kernel), models and the dataflow among the models are also delegated with composited actors [38].

Distinct from these pre-existing methods, the main contribution of this paper is to emphasize the dynamics of the model's structure and the reusability of the integrated model. The former leads to more scalable model, while the latter leads to the unified representation of the model and decouples the model from other processes (such as input and output in Ptolemy) as well.

In the following sections, a use case as an example of IEM is given to explain problems encountered by modelers in IEM practice. The unified view of IEM is proposed and the corresponding operational semantics of the integration are carefully constructed. The multidigraphic view is then given and its uniformity with the unified view is also discussed. At the end of the paper, several conclusions of our research are given.

2. A Use Case as an Example of IEM

In this section, we give a use case as an example of IEM. In the latest research, by using OMS3, Peña-Haro et al. [31] and Zhang et al. [32] integrated WOFOST, a crop growth and production model, with HYDRUS-1D, an unsaturated flow model, and MODFLOW, a saturated flow model, to simulate the interaction between crop growth and unsaturated-saturated flow processes. In the integration, the MODFLOW domain is divided into several zones according to similarities of crops, soil properties and groundwater depth. Only one WOFOST/HYDRUS-1D profile is assigned to each one of these zones, which is illustrated in Figure 1.

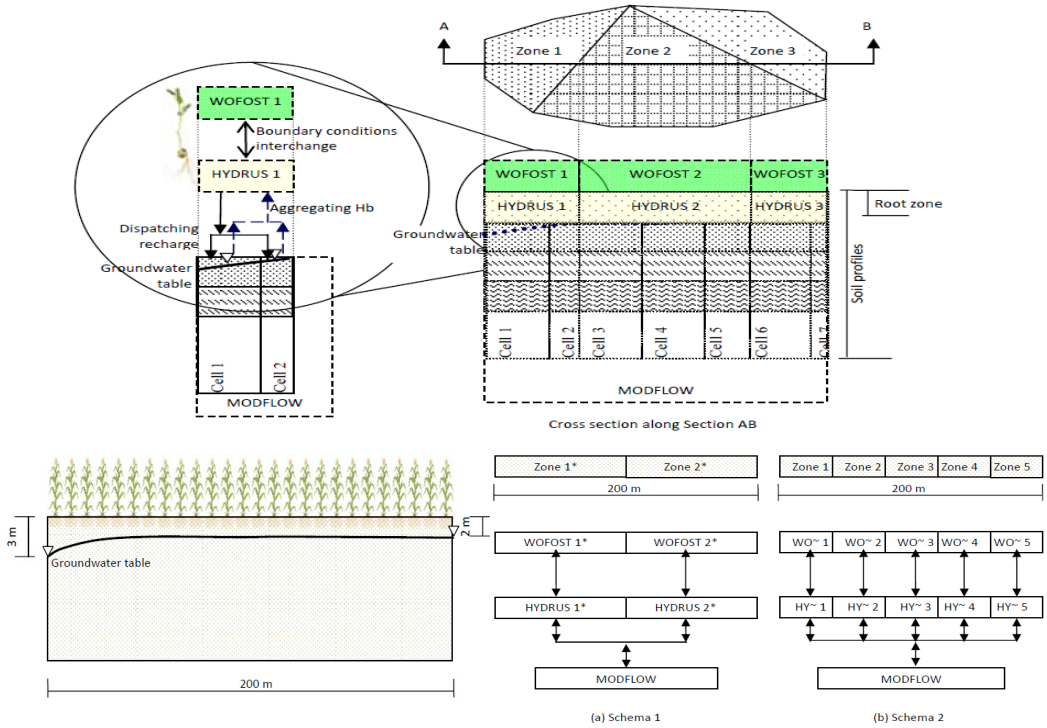


Figure 1. The conceptual model and a 2-D hypothetical simplified scenario with two spatial discretizing schemas.

In this integrated model, the WOFOST provides HYDRUS with the Leaf Area Index (LAI), root depth (RD), and crop height (CH), while HYDRUS provides WOFOST with the water stress factor (the ratio of actual transpiration (vRoot) and potential transpiration (rRoot)). Meanwhile, the HYDRUS provides MODFLOW with recharge fluxes (vBot or recharge) at the water table, while MODFLOW provides HYDRUS with the pressure head value (Hb or H) that is used as the bottom boundary condition in HYDRUS-1D. Three models' time steps are all adopted with one day. That is to say, three models simulate one day's evolution of the crop growth, unsaturated flow, and groundwater flow, respectively. The relation can be illustrated as shown in Figure 2.

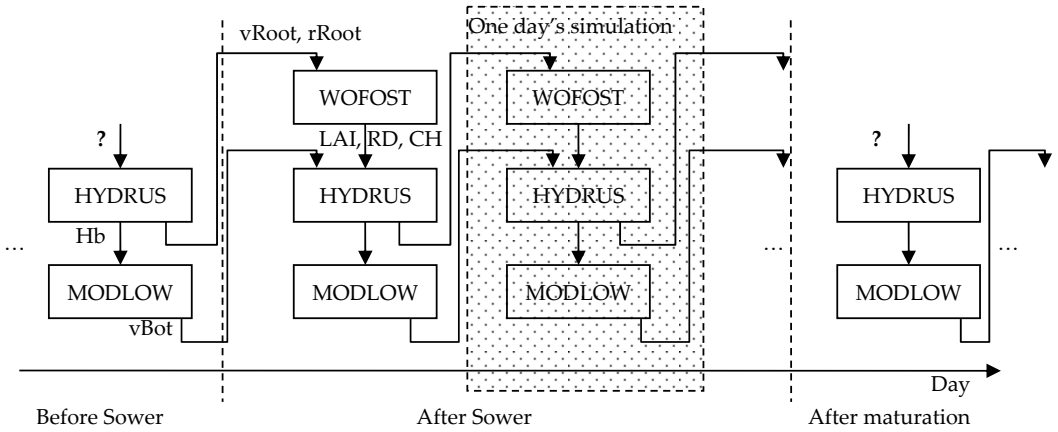


Figure 2. The process of the integrated model's simulation.

In practice, we encountered several difficulties. The first difficulty is that WOFOST has a different active period from HYDRUS1-D and MODFLOW. When there is no crop (Before sower and after maturation in Figure 2), WOFOST is deactivated and the input dataflow of HYDRUS-1D disappears.

Should we integrate only Hydrus-1D and MODFLOW, or WOFOST, HYDRUS-1D and MODFLOW together? If we adopt the former, WOFOST has to be integrated in the "after sower" period, which leads to more effort for the new integration. Moreover, the integrated model in the whole simulation is not consistent. If we adopt the later, how deal with simulation without a crop? In our earlier study, we had to add extra codes to exam the state of the crop's sower and growth and decide whether WOFOST should be triggered. At the same time, both the structure and the dataflow of the model are variable during the simulation, which makes the integrated model difficult to reuse.

The second difficulty is that the conceptually integrated model is difficult to be reuse during the instantiation. For different spatial discretizing schemas (such as Schema 1 with 2 zones and Schema 2 with 5 zones, shown in Figure 1), there are different model instances and different dataflow, so that many integrating codes have to be repeated for different schemas. Other problems will be encountered in different contexts, such as how to integrate models with different space dimensions or different time steps.

3. Unified View of the Model

The IEM support tool and runtime environment should be adaptable to different views from different users. The unified form of the model is provided to the end user by the platform to make the model easier to use. Each model's static features (parameters, input variables, and output variables involved in whole simulation) are visible to users. However, all sub-models (i.e., the blocks used to construct the complex model) of the integrated model are managed by the runtime environment, including data transfer, the models' schedules, and process control, which is transparent to the end user. During the simulation, the integrated model's parameters, input variables, and output variables may vary in different time steps. For example, in the above WOFOST, HYDRUS and MODFLOW integrated model, when the model is in the "before sowing" and "after maturation" phases, the parameters, input variables, and output variables are not needed. Therefore, the dataflow related to the model changes too. The variations of the model, its dynamic features, should be represented carefully such that the runtime environment can detect and manage them. Although FSM, Petri-Nets, and other mechanisms [35–37] can all model the dynamic features, because of the simplicity and efficiency in dataflow management [37], we chose FSM to represent the dynamic features in our framework.

In the following section, a formal framework to support these mechanisms is discussed in detail.

3.1. Formal Definition of the Model

In our framework, a unified view represents all models, no matter whether they are basic or integrated. Here the basic model implies that the model is not integrated from other models. The model is formally and abstractly defined with a 5-tuple, which is based on the concept of the actor, as

$$\mathbf{M:} = < \boldsymbol{P}, \boldsymbol{I}, \boldsymbol{O}, \boldsymbol{F}, A>$$

In this framework, $\boldsymbol{P}$, $\boldsymbol{I}$, and $\boldsymbol{O}$ are three tuples of the parameters, input variables, and output variables, respectively. They are the interfaces of the model, used to interact with the external environment. The framework defines them similarly as follows:

$$\boldsymbol{P} = \emptyset \text{ or } \boldsymbol{P} = \{p_1, p_2, \dots\}$$

$$\boldsymbol{I} = \emptyset \text{ or } \boldsymbol{I} = \{i_1, i_2, \dots\}$$

and

$$\boldsymbol{O} = \emptyset \text{ or } \boldsymbol{O} = \{o_1, o_2, \dots\}$$

$\boldsymbol{F}$ is the tuples of the functions of the model, which is defined as

$$\boldsymbol{F} = \{f_1, f_2, \dots\}.$$

The framework defines the state of the model as

$$\mathbf{S:} = < \boldsymbol{P}_e, \boldsymbol{I}_e, \boldsymbol{O}_e, f_e >$$

where $f_e \in \boldsymbol{F}$, and

$$f_e: \boldsymbol{P}_e \times \boldsymbol{I}_e \rightarrow \boldsymbol{O}_e$$

or

$$(o_{e1}, o_{e2}, \ldots, o_{en}') = f_e\,(p_{e1}, p_{e2}, \ldots, p_{el}', i_{e1}, i_{e2}, \ldots, i_{em}')$$

where $\boldsymbol{P}_e \in \boldsymbol{P}$, $\boldsymbol{I}_e \in \boldsymbol{I}$, $\boldsymbol{O}_e \in \boldsymbol{O}$, en', el', $em' \in \boldsymbol{N}$, and el', $em' \geq 0$. $\boldsymbol{P}_e$, $\boldsymbol{I}_e$, $\boldsymbol{O}_e$, and f_e are the effective parameters set, effective input variables set, effective output variables set, and effective function under the specific state, respectively.

When the model is under a specific state (such as before sowing in the example), it will have the same effective parameters set, effective input variables set, effective output variables set, and effective function set. When the new inputs are coming and they all satisfy the constraints, the model will be fired. Once $\boldsymbol{P}_e$, $\boldsymbol{I}_e$, $\boldsymbol{O}_e$, or f_e changes (such as from before sowing to after sowing), the model will change to a new state, which can be represented by the Finite States Machine (FSM).

A is a FSM of the model, which is defined as

$$\text{A:} = <\boldsymbol{S}, s_0, T>,$$

where $\boldsymbol{S}$ is the finite set of states of the model. T: $\boldsymbol{S} \rightarrow \boldsymbol{S}$ is the state transition function, which means that, while the parameter $\boldsymbol{P}_e$ and input variable $\boldsymbol{I}_e$ are valuated, the model will transit to a new state, coinciding with $\boldsymbol{P}_e$ and $\boldsymbol{I}_e$, and $\boldsymbol{f}_e$ will be fired while $\boldsymbol{O}_e$ is obtained. In the framework, the initial state $s_0 = < \emptyset, \emptyset, \emptyset, f_\emptyset >$ means that the model has been activated, where $s_0 \in \boldsymbol{S}$ and $f_\emptyset$ denotes that there is nothing to do except await new input.

In the definition, $\boldsymbol{P}$, $\boldsymbol{I}$, $\boldsymbol{O}$, and $\boldsymbol{F}$ are used to represent the static structure properties of the model. At the same time, the dynamic characteristics of the model are described with the FSM $\boldsymbol{A}$, which represents the transition of the structure of the model. We use the block diagram as shown in Figure 3 to represent the model. In the figure, the model $m = < \boldsymbol{P}, \boldsymbol{I}, \boldsymbol{O}, \boldsymbol{F}, A >$, $\boldsymbol{P} = \{p_1, p_2\}$, $\boldsymbol{I} = \{i_1, i_2, i_3\}$, $\boldsymbol{O} = \{o_1, o_2\}$, $\boldsymbol{F} = \{f_1, f_2\}$, $A = <\boldsymbol{S}, s_0, T >$, $\boldsymbol{S} = \{ s_0, s_1, s_2\}$, and $T = \{(s_0, s_1), (s_0, s_2), (s_1, s_1), (s_2, s_2),(s_1, s_2), (s_2, s_1)\}$. In addition, $s_1 = <\boldsymbol{P}_{e1}, \boldsymbol{I}_{e1}, \boldsymbol{O}_{e1}, f_1>$, $s_2 = <\boldsymbol{P}_{e2}, \boldsymbol{I}_{e2}, \boldsymbol{O}_{e2}, f_2>$, $\boldsymbol{P}_{e1} = \boldsymbol{P}_{e2} = \{p_1, p_2\}$, $\boldsymbol{I}_{e1} = \{i_1, i_2\}$, $\boldsymbol{I}_{e2} = \{i_1, i_3\}$, $\boldsymbol{O}_{e1} = \{o_1\}$, and $\boldsymbol{O}_{e2} = \{o_2\}$. The transition is illustrated in the figure.

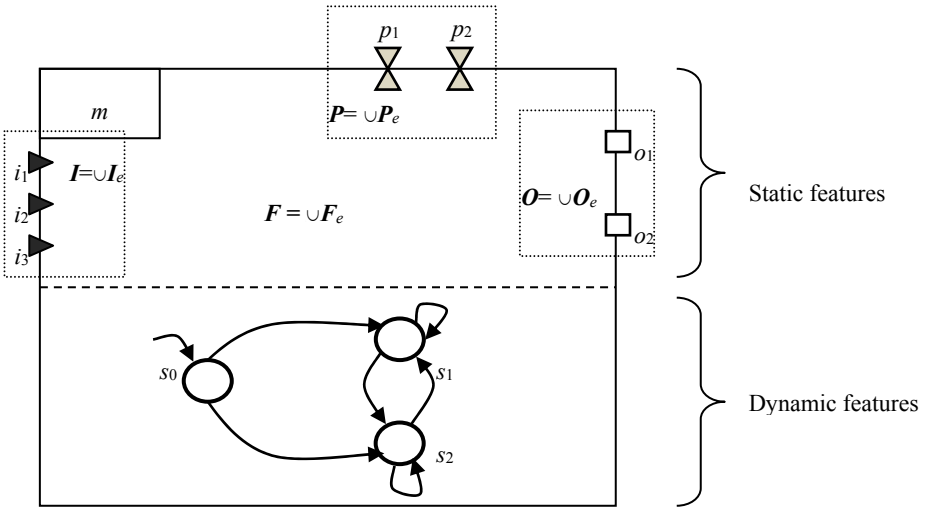


Figure 3. The diagram definition of the model.

A basic model is represented in Figure 4, where $m = <\boldsymbol{P}, \boldsymbol{I}, \boldsymbol{O}, \boldsymbol{F}, A>$, $\boldsymbol{F} = \{ f \}$, $A = <\boldsymbol{S}, s_0, T>$, $\boldsymbol{S} = \{s_0, s_1\}$, and $\boldsymbol{T} = \{ (s_0, s_1), (s_1, s_1)\}$. Additionally, $s_1 = <\boldsymbol{P}_e, \boldsymbol{I}_e, \boldsymbol{O}_e, f>$, $\boldsymbol{P}_e = \boldsymbol{P}$, $\boldsymbol{I}_e = \boldsymbol{I}$, and $\boldsymbol{O}_e = \boldsymbol{O}$.

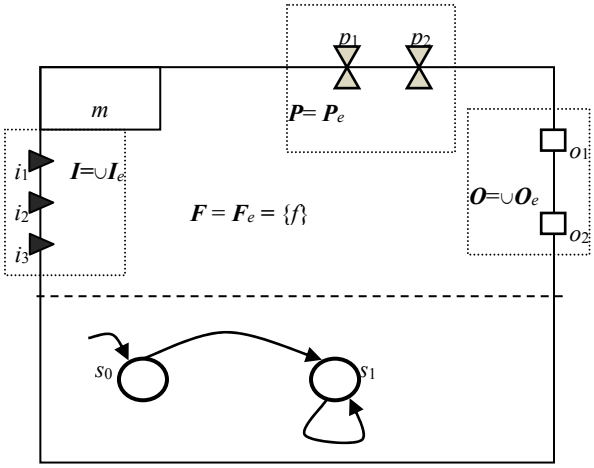


Figure 4. The diagram definition of the basic model.

3.2. Algebra for Integrating Models

To obtain the unified view of the model in integrated modeling, we present algebra like that in the works of Hamadi and Benatallah [33] to model the control flow of IM, which allows the creation of new models using the existing ones as building blocks. With this algebra, the new model also satisfies the unified definition of the model.

We describe below the syntax and informal semantics of the model algebra operators. The constructs are chosen to allow common and advanced model integration. The set of models can be defined by the following grammar in BNF-like notation:

$$M::= \varepsilon \mid X \mid M \rightarrow M \mid M \leftarrow M \mid M \Diamond M \mid M \uparrow M \mid \mu M$$

ε represents an empty model, i.e., a model that performs no operation.

X represents a model constant, used as a basic model in this context.

$M_1 \rightarrow M_2$ represents an integrated model that performs the model M_1 followed by the model M_2, i.e., $\rightarrow$ is an operator of *sequence*.

$M_1 \leftarrow M_2$ represents an integrated model where the next stage of model M_1 must be performed after the current stage of model M_2, i.e., $\leftarrow$ is the *feedback* operator.

$M_1 \Diamond M_2$ represents an integrated model that behaves as either model M_1 or model M_2. Once one of them executes its first operation, the second model is discarded; thus, $\Diamond$ is the *select* operator.

$M_1 \uparrow M_2$ represents a model in which M_1 and M_2 can execute simultaneously. There is no interaction between M_1 and M_2, i.e., $\uparrow$ represents a *parallel* operator.

μM represents a model that performs the model M a certain number of times, i.e., μ represents an *iterate* operator.

4. Formal Operational Semantics of Integrated Modeling

In the framework, the formal operational semantics are presented to represent and implement the algebra calculators for integration.

4.1. Connector

The models interact with each other through the dataflow among them. However, because the models may be developed in different scenarios for different purpose by different developers, there

can be potential gaps between models that impede them from connecting smoothly. The barriers mainly involve the different dimensions, different scales, and even different semantics between the variables that will be connected during integration.

The framework defines the connector to 'glue' the models together such that it fills in the gap. The connector is a simple and single functionality computing unit. The data from the output variable of the model is collected by the connector. A new datum is computed with those input data and sent to the input variable of another model or to the outputting model itself.

Each connector can be represented as a 5-tuple, similar to the basic model:

$$\mathbf{C:} = <\boldsymbol{P}, \boldsymbol{I}, \boldsymbol{O}, \boldsymbol{F}, A>.$$

Compared to the basic model's definition, the only difference is that $\boldsymbol{O} = \{o\}$, i.e., the connector always has only one output variable. To distinguish from the model, we use the box with a triangle to represent the connector in the block diagram, as seen in Figure 5. The value of the output variable of the first model can sometimes be transferred to the input variable of the second model directly; the connector may then be simpler, i.e., $\boldsymbol{P} = \emptyset$, $\boldsymbol{I} = \{i\}$, and f is $o = f(i) = i$.

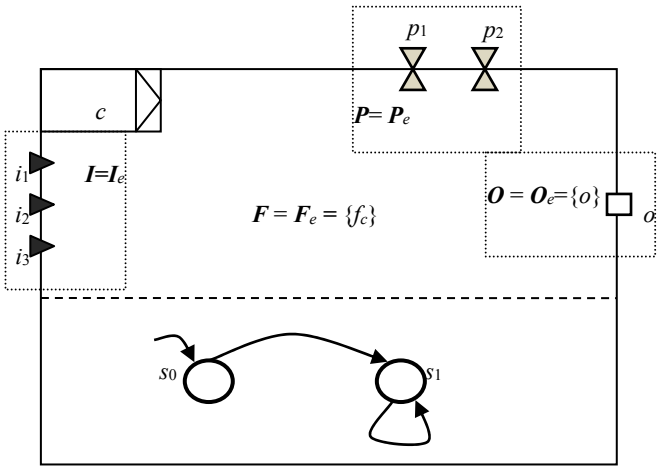


Figure 5. The diagram definition of the connector.

4.2. Semantics for the Algebraic Operator

4.2.1. Empty Model

For technical and theoretical reasons, a model that performs no operation is defined as an empty model ε.

$$\varepsilon = <\boldsymbol{P}, \boldsymbol{I}, \boldsymbol{O}, \boldsymbol{F}, A>$$

where $\boldsymbol{P} = \emptyset$, $\boldsymbol{I} = \emptyset$, $\boldsymbol{O} = \emptyset$, $\boldsymbol{F} = \emptyset$, $A = <\boldsymbol{S}, s_0, T>$, $\boldsymbol{S} = \{s_0\}$, and $T = \emptyset$.

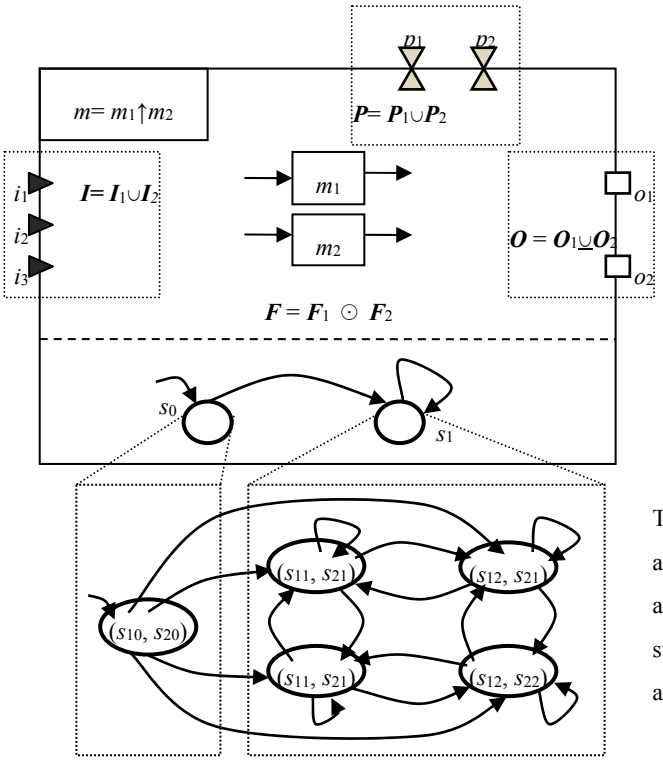
4.2.2. Parallel

The *parallel* operator indicates that two or more models can be parallel in the same cycle. Paralleled models do not depend on each other's according to the dataflow. But if one model fails, the integrated model based on the control will fail too. For example, in schema 1 of Figure 1, HYDRUS 1 and HYDRUS 2 should be integrated with the *parallel* operator. Given two models: $m_1 = <\boldsymbol{P}_1, \boldsymbol{I}_1, \boldsymbol{O}_1, \boldsymbol{F}_1, A_1>$, $m_2 = <\boldsymbol{P}_2, \boldsymbol{I}_2, \boldsymbol{O}_2, \boldsymbol{F}_2, A_2>$, $A_1 = <\boldsymbol{S}_1, s_{10}, T_1>$, and $A_2 = <\boldsymbol{S}_2, s_{20}, T_2>$, let $m = m_1 \uparrow m_2$ and $m = <\boldsymbol{P}, \boldsymbol{I}, \boldsymbol{O}, \boldsymbol{F}, A>$.

Therefore, $\boldsymbol{P} = \boldsymbol{P}_1 \cup \boldsymbol{P}_2$, $\boldsymbol{I} = \boldsymbol{I}_1 \cup \boldsymbol{I}_2$, and $\boldsymbol{O} = \boldsymbol{O}_1 \cup \boldsymbol{O}_2$, where $\cup$ implies that all output variables of the different models are different. The integration of the function can be represented with $\boldsymbol{F} = \boldsymbol{F}_1 \odot \boldsymbol{F}_2$, where $\odot$ implies $\boldsymbol{F} = \{f = (f_1 \uparrow f_2) \mid f_1 \in \boldsymbol{F}_1, f_2 \in \boldsymbol{F}_2\}$ and $f = f_1 \uparrow f_2$ implies that f executes exactly if both

f_1 and f_2 execute exactly and simultaneously. In accordance with FSM, the initial state of the model is $s_0 = <s_{10}, s_{20}>$, where s_{10} and s_{20} refer to the initial states of m_1 and m_2, respectively, which means that m_1 and m_2 are in initial states simultaneously. $\boldsymbol{S} = \{s_0\} \cup (\boldsymbol{S}_1 \backslash \{s_{10}\} \times \boldsymbol{S}_2 \backslash \{s_{20}\})$, where $\boldsymbol{S} \backslash \{s\} = (\boldsymbol{S} - \{s\})$. The transition T is also the combination of T_1 and T_2. We define $T = T_1 \times T_2 - \boldsymbol{s}_{10}\bullet - \boldsymbol{s}_{20}\bullet$, where $T_1 \times T_2 = \{(s_{1i}s_{2l}, s_{1j}s_{2m}) \mid (s_{1i}, s_{1j}) \in T_1, (s_{2l}, s_{2m}) \in T_2, i, j, l, m \geq 0\}$, $\boldsymbol{s}_{10}\bullet = \{(s_{10}s_{2l}, s_{1j}s_{2m}) \in T_1 \times T_2 \mid, j > 0, l, m \geq 0\}$, and $\boldsymbol{s}_{20}\bullet = \{(s_{1i}s_{20}, s_{1j}s_{2m}) \in T_1 \times T_2 \mid, I > 0, l, m \geq 0\}$. If $s_i = (s_{1j,\, s2k})$, $\boldsymbol{P}_{ei} = \boldsymbol{P}_{e1j} \cup \boldsymbol{P}_{e2k}$, $\boldsymbol{I}_{ei} = \boldsymbol{I}_{e1j} \cup \boldsymbol{I}_{e2k}$, $\boldsymbol{O}_{ei} = \boldsymbol{O}_{e1j} \cup \boldsymbol{O}_{e2k}$, $i \neq 0$, and $f_e = (f_{e1} \uparrow f_{e2})$, which means f_{e1} and f_{e2} are performed simultaneously. The parallel operation is illustrated in Figure 6.

It is obvious that we can obtain $m = m_1 \uparrow m_2 = m_2 \uparrow m_1$ and $m = m_1 \uparrow m_2 \uparrow m_3 = (m_1 \uparrow m_2) \uparrow m_3 = m_1 \uparrow (m_2 \uparrow m_3)$.



To clarify, we consider that m_1 and m_2 both have two states in addition to their respective initial states. These states are denoted as s_{11}, s_{12} of m_1 and s_{21}, s_{22} of m_2.

Figure 6. The diagram definition of the *Parallel* operator.

4.2.3. Sequence

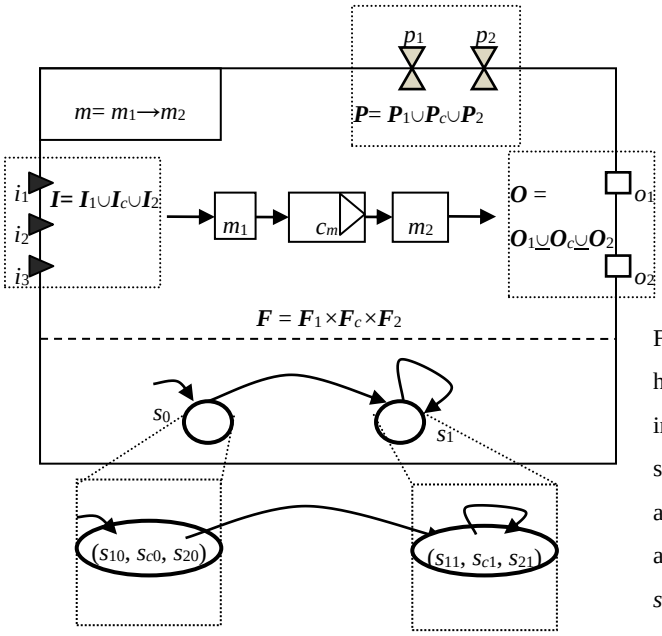
The *sequence* operator is the most familiar operation during integration. It indicates that m_1 must have finished its action before m_2 can be activated in the same cycle, which is generally caused by the dataflow. If the data of one or more input variables of m_2 are from the output variables of m_1, then there is a sequence $m_1 \rightarrow m_2$. Usually, there is one or more connectors between m_1 and m_2; we denote this as $m_1 \rightarrow (c_1 \uparrow \ldots \uparrow c_k) \rightarrow m_2$, where $c_1, \ldots,$ and c_k are parallel.

Suppose there are two models $m_1 = <\boldsymbol{P}_1, \boldsymbol{I}_1, \boldsymbol{O}_1, \boldsymbol{F}_1, A_1>$, $m_2 = <\boldsymbol{P}_2, \boldsymbol{I}_2, \boldsymbol{O}_2, \boldsymbol{F}_2, A_2>$, $A_1 = <\boldsymbol{S}_1, s_{10}, T_1>$, and $A_2 = <\boldsymbol{S}_2, s_{20}, T_2>$. If $m = <\boldsymbol{P}, \boldsymbol{I}, \boldsymbol{O}, \boldsymbol{F}, A>$ and $m = m_1 \rightarrow m_2 = m_1 \rightarrow (c_1 \uparrow \ldots \uparrow c_k) \rightarrow m_2$, then, according to parallel operation, we can define $c_m = c_1 \uparrow \ldots \uparrow c_k$, and $c_m = <\boldsymbol{P}_c, \boldsymbol{I}_c, \boldsymbol{O}_c, \boldsymbol{F}_c, A_c>$, $A_c = <\boldsymbol{S}_c, s_{c0}, T_c>$, $\boldsymbol{S}_c = \{s_{c0}, s_{c1}\}$, and $T_c = \{<s_{c0}, s_{c1}>, <s_{c1}, s_{c1}>\}$.

As a result, we can obtain $\boldsymbol{P} = \boldsymbol{P}_1 \cup \boldsymbol{P}_c \cup \boldsymbol{P}_3$, $\boldsymbol{O} = \boldsymbol{O}_1 \cup \boldsymbol{O}_c \cup \boldsymbol{O}_3$, and $\boldsymbol{I} = \boldsymbol{I}_1 \cup \boldsymbol{I}_c \cup \boldsymbol{I}_2 - \boldsymbol{I}_{c\hat{}1} - \boldsymbol{I}_{2\hat{}c}$, where $\boldsymbol{I}_{a\hat{}b} = \{i \in \boldsymbol{I}_a \mid \exists o \in \boldsymbol{O}_b, m_b.o = m_a.i>\}$ is the set of m_b's inputs that have been occupied by m_a's outputs.

The integration of the function can be represented with $\boldsymbol{F} = \boldsymbol{F}_1 \times \boldsymbol{F}_c \times \boldsymbol{F}_2$, and $f = (f_1 \rightarrow f_c \rightarrow f_2) \in \boldsymbol{F}$, which means that $f_1 \in \boldsymbol{F}_1$, $f_c \in \boldsymbol{F}_c$, and $f_2 \in \boldsymbol{F}_2$ are performed in sequence. According to FSM, $A = A_1 \times A_c \times A_2$, and the initial state of the model is $s_0 = <s_{10}, s_{c0}, s_{20}>$, which means that m_1, c and m_2 are

in initial states simultaneously. In addition, $\boldsymbol{S} = \{s_0\} \cup (\boldsymbol{S}_1 \backslash \{s_{10}\} \times \boldsymbol{S}_c \backslash \{s_{c0}\} \times \boldsymbol{S}_2 \backslash \{s_{20}\})$. The transition T is also the combination of T_1, T_c and T_2. We define $T = T_1 \times T_c \times T_2 - \boldsymbol{s}_{10}\bullet - \boldsymbol{s}_{c0}\bullet - \boldsymbol{s}_{20}\bullet$. If $s_i = (s_{1j}, s_{c1}, s_{2k})$, then $\boldsymbol{P}_{ei} = \boldsymbol{P}_{e1j} \cup \boldsymbol{P}_{ec1} \cup \boldsymbol{P}_{e2k}$, $\boldsymbol{I}_{ei} = \boldsymbol{I}_{ej1} \cup \boldsymbol{I}_{ecl} \cup \boldsymbol{I}_{e2k} - \boldsymbol{I}_{ecl \hat{} e1j} - \boldsymbol{I}_{e2kl \hat{} ecl}$, $\boldsymbol{O}_{ei} = \boldsymbol{O}_{e1j} \cup \boldsymbol{O}_{ec1} \cup \boldsymbol{O}_{e2k}$, $i \neq 0$, $f_e = (f_{e1} \rightarrow f_{ec} \rightarrow f_{e2})$. The sequence operation is illustrated in Figure 7.



For simplicity, m_1 and m_2 both have one state besides their initial respective states. These states are denoted as s_{11} of m_1 and s_{21} of m_2. Connector c_m always has two states: s_{c0} and s_{c1}.

Figure 7. The diagram definition of the *Sequence* operator.

We can also obtain $m = m_1 \rightarrow m_2 \rightarrow m_3 = (m_1 \rightarrow m_2) \rightarrow m_3 = m_1 \rightarrow (m_2 \rightarrow m_3)$. In our example model, WOFOST to HYDRUS-1D and HYDRUS-1D to MODFLOW are sequence relations, which can be represented as WOFOST → HYDRUS-1D and HYDRUS-1D → MODFLOW, respectively. Together these are equal to WOFOST → HYDRUS-1D → MODFLOW. The parameters of the integrated model are the union of three sub-models' parameters. The input variables should be the union of three sub-models' variables except LAI, RD, CH (three HYDRUS-1D's input variables) and Hb (a WOFOST's input variable). The output variables are the union of three models' output variables. The function of the integrated model is the combination of three sub-models. In the model, there are only two states: $s_0 = (s_{10}, s_{20}, s_{30})$ (initial states) and $s_1 = (s_{11}, s_{21}, s_{31})$.

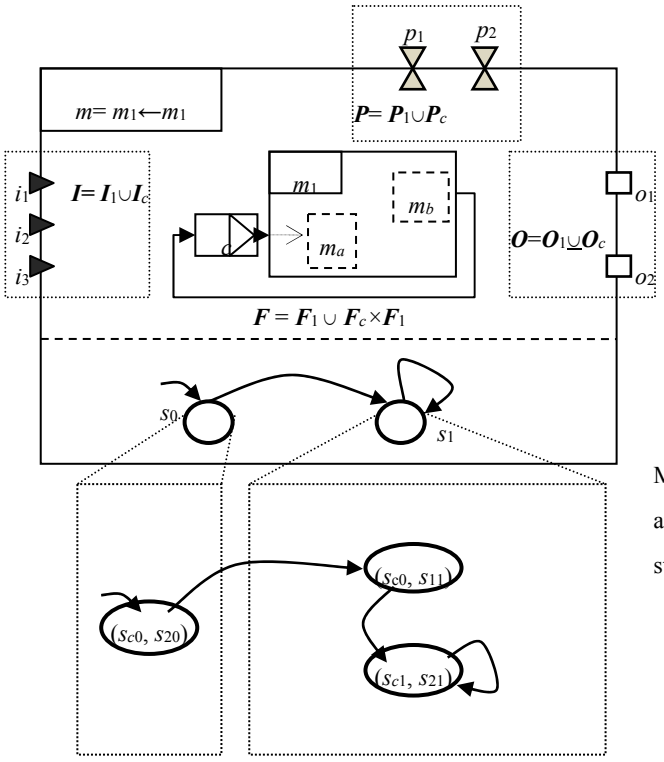
Sometimes, there may be no explicit dataflow between two models. In a cycle, the successor models can begin to run if and only if the predecessor models have finished. The empty connector c_ε, which is similar to the empty model, can be used to imply these sequences.

4.2.4. Feedback

A feedback operation occurs across two cycles of the model. It may act on one model or on two different models, denoted with $m_1 \leftarrow m_1$ or $m_1 \leftarrow m_2$. This implies that the data are transferred from an output variable of the later model in the previous cycle to the input variable of the former at the next cycle through a connector, c. The second type of feedback operation cannot exist on its own. Generally, it will be based on a sequence or parallel operation in a single cycle. Additionally, it is obvious that m_1 and m_2 can be understood as a model m's different components, such that $m_1 \leftarrow m_2$ is equivalent to $m \leftarrow m$. Thus, we need only to discuss the semantics of $m_1 \leftarrow m_1$ or $(c \rightarrow m_1) \leftarrow m_1$.

Suppose $m_1 = <\boldsymbol{P}_1, \boldsymbol{I}_1, \boldsymbol{O}_1, \boldsymbol{F}_1, A_1>$, $A_1 = <\boldsymbol{S}_1, s_{10}, T_1>$, $c = <\boldsymbol{P}_c, \boldsymbol{I}_c, \boldsymbol{O}_1, \boldsymbol{F}_c, A_c>$, $A_c = <\boldsymbol{S}_c, s_{c0}, T_c>$. Let $m = (c \rightarrow m_1) \leftarrow m_1$ and $m = <\boldsymbol{P}, \boldsymbol{I}, \boldsymbol{O}, \boldsymbol{F}, A>$, $A = <\boldsymbol{S}, s_0, T>$.

We can thus obtain the following conclusions. $\boldsymbol{P} = \boldsymbol{P}_1 \cup \boldsymbol{P}_c$, $\boldsymbol{I} = \boldsymbol{I}_1 \cup \boldsymbol{I}_c$, $\boldsymbol{O} = \boldsymbol{O}_1 \cup \boldsymbol{O}_c$, $\boldsymbol{F} = \boldsymbol{F}_c \times \boldsymbol{F}_1 = \{f_c\} \times \boldsymbol{F}_1$, and $A = A_c \times A_1$. Further, $s_0 = (s_{c0}, s_{10})$, $\boldsymbol{S} = \{s_0\} \cup \boldsymbol{S}_c \backslash \{s_{c0}\} \times \boldsymbol{S}_1 \backslash \{s_{10}\}$ and $T = \{(s_{c0}, s_{10}), (s_{c0}, s_{11})\} \cup \boldsymbol{s}_{c1}\bullet$. If $t \in \{(s_{c0}, s_{10}), (s_{c0}, s_{11})\}$, then the corresponding target state $s_t = <\boldsymbol{P}_e, \boldsymbol{I}_e, \boldsymbol{O}_e, f_e>$, $\boldsymbol{P}_{est} = \boldsymbol{P}_{ec} \cup \boldsymbol{P}_{e1}$, $\boldsymbol{I}_{est} = \boldsymbol{I}_{ec} \cup \boldsymbol{I}_{e1}$, $\boldsymbol{O}_{est} = \boldsymbol{O}_{ec} \cup \boldsymbol{O}_{e1}$, $f_e = f_c \rightarrow f_1$. Otherwise, the formulae are the same as before, except that $\boldsymbol{I}_{est} = \boldsymbol{I}_{ec} \cup \boldsymbol{I}_{e1}$ - $\{i'\}$, where i' is transferred from the last cycle's output. The feedback operation is illustrated in Figure 8. For our example model, WOFOST ← HYDRUS-1D and HYDRUS-1D ← MODFLOW are the feedback relation. They also can be represented as cascade formation WOFOST ← HYDRUS-1D ← MODFLOW. The detail of this feedback is omitted.



Model m_1 has the states s_{10}, s_{11}, and s_{12}. Connector c has two states: s_{c0}, and s_{c1}.

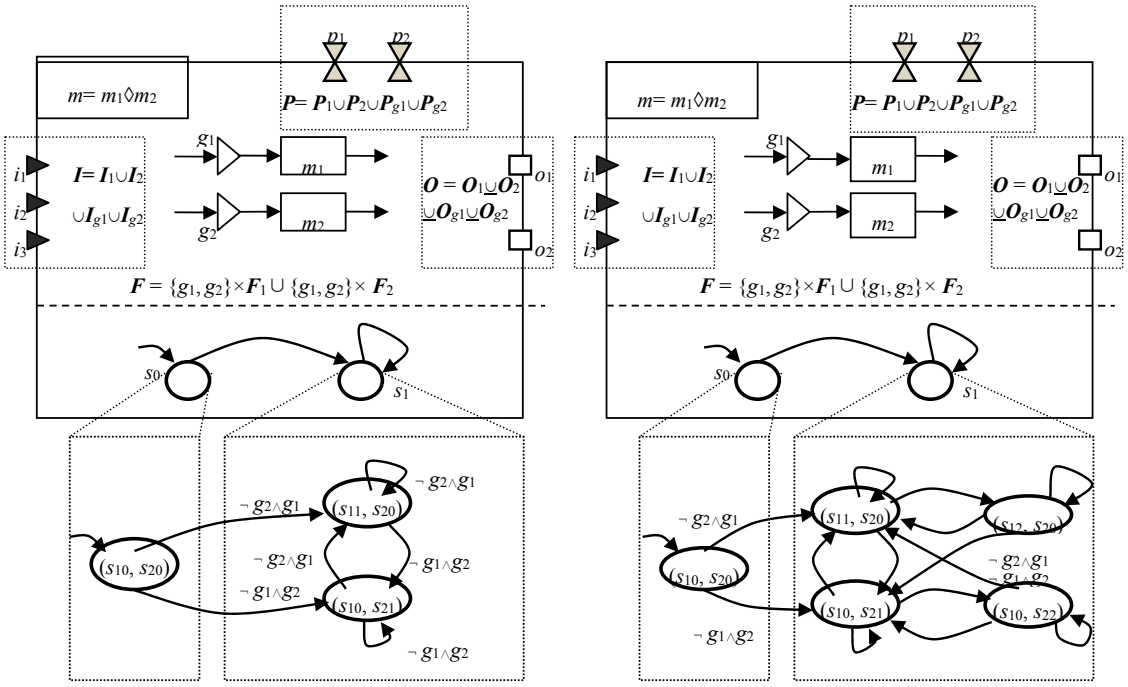
Figure 8. The diagram definition of the *Feedback* operator.

4.2.5. Select

The *Select* operation makes choosing between different sub-models or sub-model groups possible. These sub-models or sub-model groups may have similar functions and different implementation methods, or face different initial or boundary conditions. For example, in the integration of HYDRUS and WOFOST, before the crop sprouts and after the crop harvests, WOFOST need not work all the time.

Suppose $m_1 = <\boldsymbol{P}_1, \boldsymbol{I}_1, \boldsymbol{O}_1, \boldsymbol{F}_1, A_1>$, $m_2 = <\boldsymbol{P}_2, \boldsymbol{I}_2, \boldsymbol{O}_2, \boldsymbol{F}_2, A_2>$, $A_1 = <\boldsymbol{S}_1, s_{10}, T_1>$, and $A_2 = <\boldsymbol{S}_2, s_{20}, T_2>$. If $m = <\boldsymbol{P}, \boldsymbol{I}, \boldsymbol{O}, \boldsymbol{F}, A>$ and $m = m_1 \Diamond m_2$, this implies that if the guard expression g_1 is satisfied, then m_1 is performed, or if g_2 is satisfied, then m_2 is performed. In *Select* operation, each guard expression has a parameter set $\boldsymbol{P}_g$, input variable set $\boldsymbol{I}_g$, and output variable set $\boldsymbol{O}_g$. We can define $\boldsymbol{P} = \boldsymbol{P}_1 \cup \boldsymbol{P}_2 \cup \boldsymbol{P}_{g1} \cup \boldsymbol{P}_{g2}$, $\boldsymbol{I} = \boldsymbol{I}_1 \cup \boldsymbol{I}_2 \cup \boldsymbol{I}_{g1} \cup \boldsymbol{I}_{g2}$, $\boldsymbol{O} = \boldsymbol{O}_1 \cup \boldsymbol{O}_2 \cup \boldsymbol{O}_{g1} \cup \boldsymbol{O}_{g2}$, and $\boldsymbol{F} = \{g_1, g_2\} \times \boldsymbol{F}_1 \cup \{g_1, g_2\} \times \boldsymbol{F}_2$. For the FSM A, $\boldsymbol{S} = \{s_{10}\} \times \boldsymbol{S}_2 \cup \boldsymbol{S}_1 \times \{s_{20}\}$, $s_0 = \{s_{10}, s_{20}\}$, and $T = \cup^k{}_{i=0}(T_1 \times T_{2i0}) \cup (\cup^k{}_{i=0}(T_{1i0} \times T_2))$, where T_{1i0} and T_{2i0} are not existing transitions that imply (s_{1i}, s_{10}) for model m_1 and (s_{2i}, s_{20}) for model m_2, and where s_{1i} and s_{2i} stand for arbitrary states of m_1 and m_2, respectively. Suppose $s_i = <\boldsymbol{P}_e, \boldsymbol{I}_e, \boldsymbol{O}_e, f_e>$. If $s_i = (s_{1j}, s_{20})$, then $\boldsymbol{P}_e = \boldsymbol{P}_{e1j} \cup \boldsymbol{P}_{g1} \cup \boldsymbol{P}_{g2}$, $\boldsymbol{I}_e = \boldsymbol{I}_{e1j} \cup \boldsymbol{I}_{g1} \cup \boldsymbol{I}_{g2}$, $\boldsymbol{O}_e = \boldsymbol{O}_{e1j} \cup \boldsymbol{O}_{g1} \cup \boldsymbol{O}_{g2}$, $f_e = (f_{g1}, f_{g2})$ & f_{e1j}. If $s_i = (s_{10}, s_{2j})$, then $\boldsymbol{P}_e = \boldsymbol{P}_{e2j} \cup \boldsymbol{P}_{g1} \cup \boldsymbol{P}_{g2}$, $\boldsymbol{I}_e = \boldsymbol{I}_{e2j} \cup \boldsymbol{I}_{g1} \cup \boldsymbol{I}_{g2}$, $\boldsymbol{O}_e = \boldsymbol{O}_{e2j} \cup \boldsymbol{O}_{g1} \cup \boldsymbol{O}_{g2}$, $f_e = (g_1, g_2)$ & f_{e2j}. The select operation is illustrated in Figure 9.

Similarly, we can obtain $m = m_1 \Diamond m_2 = m_2 \Diamond m_1$ and $m = m_1 \Diamond m_2 \Diamond m_3 = (m_1 \Diamond m_2) \Diamond m_3 = m_1 \Diamond (m_2 \Diamond m_3)$. In the example, WOFOST is integrated with select operation WOFOST $\Diamond \varepsilon$.



a. $S_1 = \{s_{10}, s_{11}\}$, $S_2 = \{s_{20}, s_{21}\}$ b. $S_1 = \{s_{10}, s_{11}, s_{12}\}$, $S_2 = \{s_{20}, s_{21}, s_{22}\}$

Figure 9. The diagram definition of the *Select* operator. (**a**) S1 = {s_{10}, s_{11}}, S2 = {s_{20}, s_{21}}; (**b**) S1 = {s_{10}, s_{11}, s_{12}}, S2 = {s_{20}, s_{21}, s_{22}}.

4.2.6. Iterate

In environmental simulations, different time steps must be permitted, such as, for example, from hourly simulation (e.g., to simulate a fungal infestation) up to several decades for sustainability studies.

Although most models work iteratively, the framework only defines the iterate operation that is used when the model needs to perform many cycles while other models that will be integrated only perform one cycle. The typical scenario is when models with different time steps are to be integrated: the model with the shorter time step should iterate many times while the other model only executes once. In integrated algebra, we use μm to denote a model's iteration.

Consider the model $m_1 = <\boldsymbol{P}_1, \boldsymbol{I}_1, \boldsymbol{O}_1, \boldsymbol{F}_1, A_1>$ and $A_1 = <\boldsymbol{S}_1, s_{10}, T_1>$. If $m = <\boldsymbol{P}, \boldsymbol{I}, \boldsymbol{O}, \boldsymbol{F}, A>$ and $m = \mu m_1$, which imply m_1 is performed until the guard expression g is not satisfied (g has parameter set $\boldsymbol{P}_g$, input variable set $\boldsymbol{I}_g$, and output variable set $\boldsymbol{O}_g$), then we can obtain $\boldsymbol{P} = \boldsymbol{P}_g \cup \boldsymbol{P}_1$, $\boldsymbol{I} = \boldsymbol{I}_g \cup \boldsymbol{I}_1$, $\boldsymbol{O} = \boldsymbol{O}_g \sqcup \boldsymbol{O}_1$, and $\boldsymbol{F} = \{g\} \times \boldsymbol{F}_1 \cup \{g\} \times \{f_\varepsilon\}$.

For the FSM $A = <\boldsymbol{S}, s_0, T>$, we define $s_0 = s_{10}$. For convenience when integrating with other operations, the running state of the model is extended to comprise three types of states: the first running state $s_f \in \boldsymbol{S}_f$, where $s_f = \{\boldsymbol{P}_{ef}, \boldsymbol{I}_{ef}, \boldsymbol{O}_{ef}, f_{ef}\}$; the iterative running state $\boldsymbol{S}_{it}$; and the state of end iteration s_e, such that $\boldsymbol{S} = \{s_0, s_e\} \cup \boldsymbol{S}_f \cup \boldsymbol{S}_{it}$ and the transitions according to these states are T_f, T_{it}, and T_e, respectively. If $(s_{10}, s_{1i}) \in T_f$, then $(s_{10}, s_{1i}) \in T_1$. Let $s_{1i} = <\boldsymbol{P}_{e1i}, \boldsymbol{I}_{e1i}, \boldsymbol{O}_{e1i}, f_{e1i}>$; then $\boldsymbol{P}_{ef} = \boldsymbol{P}_{e1i} \cup \boldsymbol{P}_g$, $\boldsymbol{I}_{ef} = \boldsymbol{I}_{e1i} \cup \boldsymbol{I}_g$, $\boldsymbol{O}_{ef} = \boldsymbol{O}_{e1i} \sqcup \boldsymbol{O}_g$, and $f_{ef} = g \times f_{e1i}$. For each $(s_{1i}, s_{1j}) \in T_1$, $I \neq 0$, $(s_{1i}, s_{1j}) \in T_{it}$. Let $s_{it} = <\boldsymbol{P}_{eit}, \boldsymbol{I}_{eit}, \boldsymbol{O}_{eit}, f_{eit}>$; then, $\boldsymbol{P}_{eit} = \boldsymbol{P}_{e1j} \cup \boldsymbol{P}_g$, $\boldsymbol{I}_{eit} = \boldsymbol{I}_{e1j} \cup \boldsymbol{I}_g$, $\boldsymbol{O}_{eit} = \boldsymbol{O}_{e1j} \sqcup \boldsymbol{O}_g$, and $f_{eit} = g \times f_{e1j}$. For the end iteration state, let $s_e = <\boldsymbol{P}_{ee}, \boldsymbol{I}_{ee}, \boldsymbol{O}_{ee}, f_{ee}>$, $\boldsymbol{P}_{ee} = \boldsymbol{P}_g$, $\boldsymbol{I}_{ee} = \boldsymbol{I}_g$, $\boldsymbol{O}_{ee} = \boldsymbol{O}_g$, and $f_{ee} = g$. Then, $Te = \{(s_i, s_e) \mid s_i \in \boldsymbol{S}_f \cup \boldsymbol{S}_{it}\}$. Figure 10 illustrates the iterate operation of the model.

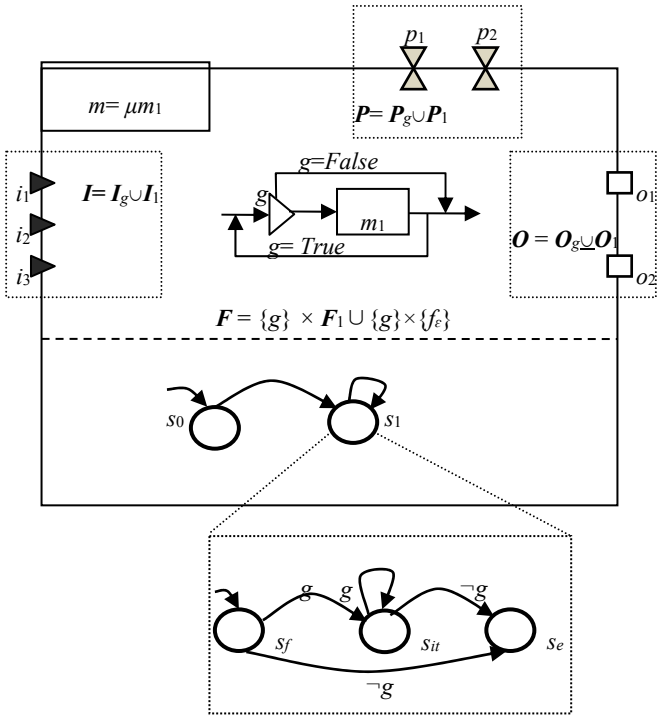


Figure 10. The diagram definition of the *Iterate* operator.

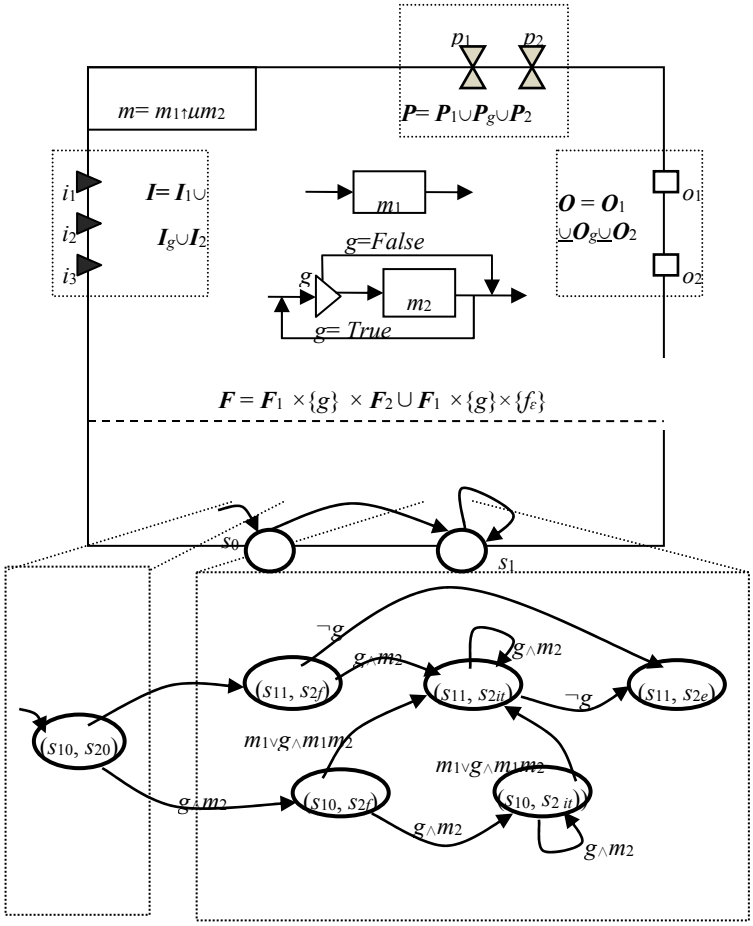
4.2.7. Combine Iterate with Other Operators

Because the iterate operation should not be used alone, it is combined with the parallel, sequence, feedback, and select operations. The framework uses $M{\uparrow}\mu M$, $M{\rightarrow}\mu M$, $\mu M{\rightarrow}M$, $\mu M{\leftarrow}M$, $M{\leftarrow}\mu M$, and $M{\Diamond}\mu M$ to denote these combinations.

$M{\uparrow}\mu M$ represents a model that parallels an iteration of another model. Let $m = m_1{\uparrow}\mu m_2$, i.e., model m_1 parallels an iteration of model m_2. Obviously, if the iteration is performed n times, then it can be extended to n cycles in which $m_1{\uparrow}m_2$ is only performed in one cycle, while in any other cycles, $\varepsilon{\uparrow}m_2$ is performed, such that $\boldsymbol{P} = \boldsymbol{P}_1 \cup \boldsymbol{P}_{m2}$, $\boldsymbol{I} = \boldsymbol{I}_1 \cup \boldsymbol{I}_{m2}$, $\boldsymbol{O} = \boldsymbol{O}_1 \cup \boldsymbol{O}_{m2}$, and $\boldsymbol{F} = \boldsymbol{F}_1 \cup \boldsymbol{F}_{m2}$. For FSM A, $S = \{s_0, s_f, s_f', s_{it}, s'_{it}, s_e\}$, $s_0 = (s_{10}, s_{20})$, $s_f = (s_{10}, s_{2f})$, $s_f' = (s_{11}, s_{2f})$, $s_{it} = (s_{10}, s_{2it})$, $s'_{it} = (s_{11}, s_{2it})$, snf $s_e = (s_{11}, s_{2e})$. $T = T_1 \times T_2 - \{((s_{10}, s_{2f}), (s_{11}, s_{2f})), ((s_{10}, s_{2it}),(s_{10}, s_{2e}))\}$. Additionally, from state (s_{10}, s_{20}) to state (s_{11}, s_{2e}), m_1 should execute only once and m_2 should execute once at least. The integration is illustrated in Figure 11.

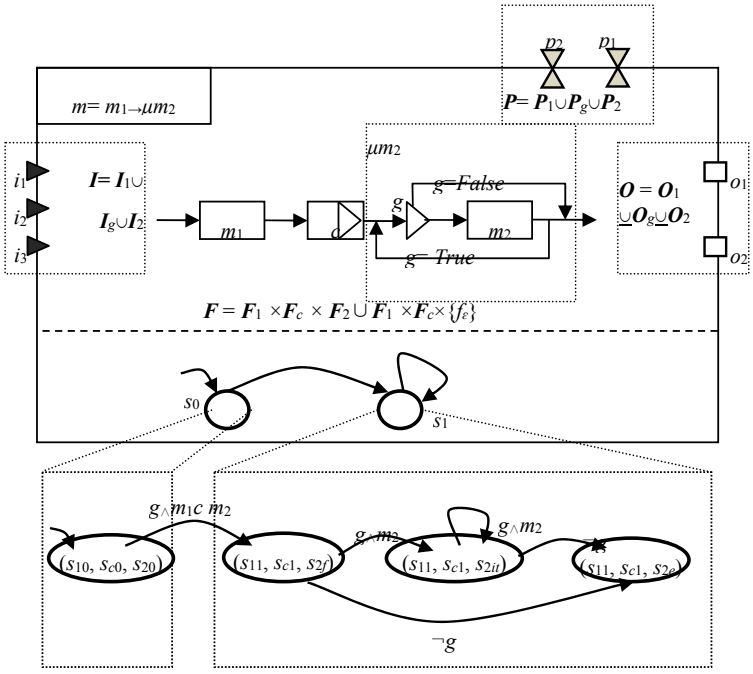
$M{\rightarrow}\mu M$ and $\mu M{\rightarrow}M$ imply that a model's iteration performs with another model in sequence. They can also be extended as what is defined in the combination of parallel and iteration. The former is extended to a sequence $M{\rightarrow}M$ in the first cycle and a series of cycles involving $\varepsilon{\rightarrow}M$. Similarly, the latter is extended to a series of cycles involving $\varepsilon{\rightarrow}M$ first and then a sequence $M{\rightarrow}M$ in the last cycle. The integration is illustrated in Figure 12.

The combination of feedback with iteration is similar to the sequence. The combination of select with iteration is simple as well. Thus, these combinations are omitted in this paper.



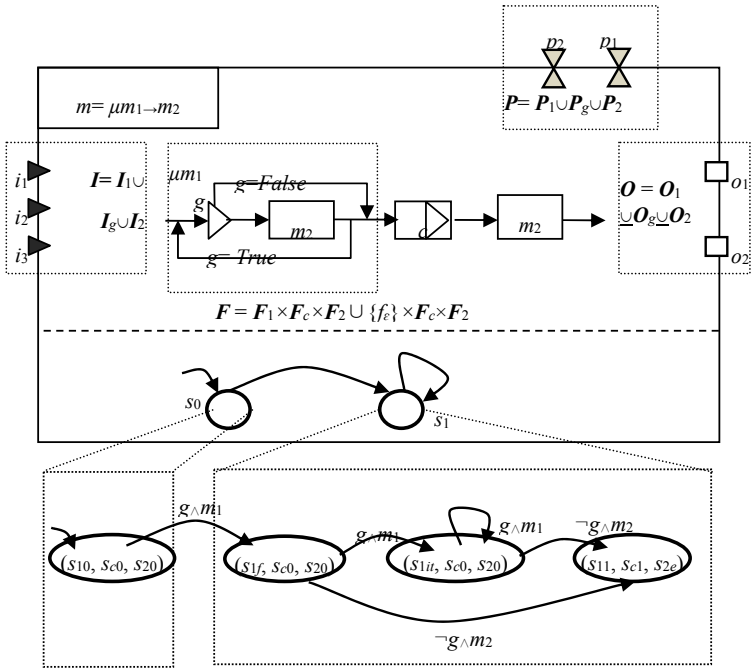
From state (s_{10}, s_{20}) to state (s_{11}, s_{2e}), m_1 should execute only once and m_2 should execute at least once.

Figure 11. The diagram definition of $M\uparrow\mu M$.



(**a**)

Figure 12. *Cont*.



(**b**)

Figure 12. The diagram definition of $M \rightarrow \mu M$. (**a**) $M \rightarrow Mm$; (**b**) $\mu M \rightarrow M$.

5. Complex Integrated Model

5.1. Model Graph

With these basic operators, a complex model can be integrated from simpler models, even basic models. A result of an *Iterate* operation or *Select* operation is taken as a sub-model in the frame. From the perspective of the integrator, many sub-models can be synthetized together with connectors and corresponding message transferring channels. These sub-models may involve both basic models and integrated models, such that the sub-models and channels compose a graph called a model graph. The model graph can be illustrated as in Figure 13, which is a simple example.

5.2. Formal Semantics of Model Graph

Given three finite alphabets ΣM, ΣC, and ΣVar as the available labels for the models, connectors and variables, respectively, the integration of the models is defined with a labeled multidigraph with ports, as follows:

$$\mathbf{G}_M: = <\boldsymbol{M}, \boldsymbol{C}, \boldsymbol{L}, \boldsymbol{PVa}, \boldsymbol{IVar}, \boldsymbol{OVar}, \boldsymbol{s}, \boldsymbol{t}, \iota, \boldsymbol{l}M, \boldsymbol{l}C, \boldsymbol{l}Var>.$$

$\boldsymbol{M}$ and $\boldsymbol{C}$ are two finite sets of vertices that denote models and connectors, respectively. $\boldsymbol{M}$ includes basic models and pre-existing integrated models.

$\boldsymbol{L} \subseteq (\boldsymbol{M} \times \boldsymbol{OVar}) \times (\boldsymbol{C} \times \boldsymbol{IVar}) \cup (\boldsymbol{C} \times \boldsymbol{OVar}) \times (\boldsymbol{M} \times \boldsymbol{IVar})$ is a set of direct edges, which indicates the dataflow among the sub-models and connectors. The edge set includes both sequence and feedback related dataflow. To distinguish them, we use the label l_s for sequence edge and l_b for feedback edge, and $\boldsymbol{L} = \boldsymbol{L}_s \cup \boldsymbol{L}_b$.

$\boldsymbol{P}$, $\boldsymbol{I}$, and $\boldsymbol{O}$ are the set of parameters, input variables, and output variables, respectively, and $\boldsymbol{Var} = \boldsymbol{P} \cup \boldsymbol{I} \cup \boldsymbol{O}$.

$\boldsymbol{\iota} = \boldsymbol{\iota_M} \cup \boldsymbol{\iota_C}$. $\boldsymbol{\iota_M} = (\iota p, \iota i, \iota o)$: $\boldsymbol{M} \rightarrow PVar \times IVar \times OVar$ assigns a parameter variable (port) set, an input variable (port) set and an output variable (port) set to each model, with $\iota p(M) = \boldsymbol{P}M$, $\iota i(M) = \boldsymbol{I}M$, $\iota o(M) = \boldsymbol{O}M$, and $\iota(M) = (\boldsymbol{P}M, \boldsymbol{I}M, \boldsymbol{O}M)$.

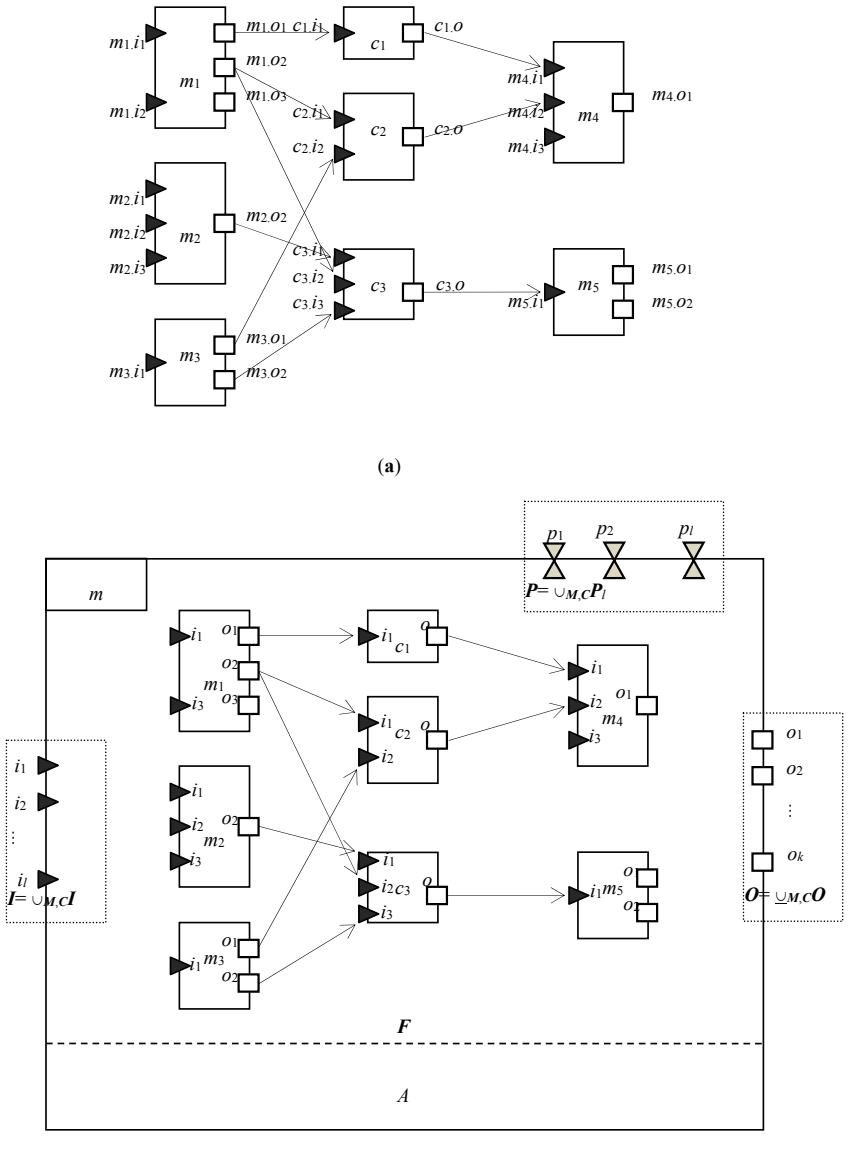


Figure 13. The example of a model graph. (**a**) The example of a model graph; (**b**) The unified view of an integrated model based on a.

Similarly, $\boldsymbol{\iota_C} = (\iota p, \iota i, \iota o)$: $\boldsymbol{C} \rightarrow \boldsymbol{P}Var \times \boldsymbol{I}Var \times \boldsymbol{O}Var$ assigns a parameter variable (port) set, an input variable (port) set and an output variable (port) set to each connector, with $\iota p(C) = \boldsymbol{P}C$, $\iota i(C) = \boldsymbol{I}C$, $\iota o(C) = \boldsymbol{O}C$, and $\iota(C) = (\boldsymbol{P}C, \boldsymbol{I}C, \boldsymbol{O}C)$.

s: $L \rightarrow \boldsymbol{M} \times \boldsymbol{O}Var \cup \boldsymbol{C} \times \boldsymbol{O}Var$ and ***t:*** $L \rightarrow \boldsymbol{M} \times \boldsymbol{I}Var \cup \boldsymbol{C} \times \boldsymbol{I}Var$ are two maps indicating the *source* and *target* model/connector of an edge. A map $\boldsymbol{n}$: $\boldsymbol{M} \times \boldsymbol{O}Var \cup \boldsymbol{C} \times \boldsymbol{O}Var \cup \boldsymbol{M} \times \boldsymbol{O}Var \cup \boldsymbol{C} \times \boldsymbol{O}Var \rightarrow \{0\} \cup \boldsymbol{N}$ is defined for the source and target of the edge to indicate the produce rate and consumption rate, respectively.

$\boldsymbol{l}M$: $\boldsymbol{M} \rightarrow \Sigma M$, $\boldsymbol{l}C$: $\boldsymbol{C} \rightarrow \Sigma C$ and $\boldsymbol{l}Var$: $\boldsymbol{L} \rightarrow \Sigma Var$ are three maps describing the labeling of the models, connectors and variables.

5.3. Construct Complex Integrated Model

The model graph is intuitive, such that the integrator can use it easily to represent the interaction of integrated models. At the same time, the IEM tools can use the properties of the graph to check the

integrated schema. In running time, the structure of the graph can be used to monitor the procedure of the simulation. Once errors occur, the IEM tool can locate them with the graph as well.

In the graph model, there are two types of relations between any two sub-models. The first is the partial ordering relation, which is based on the sequence operator. If there is a path between two sub-models m_h and m_t such that $m_h \rightarrow m_1$, $m_1 \rightarrow m_2$, ... , $m_i \rightarrow m_t$, we can say that $m_h \rightarrow\rightarrow m_t$ and call it order. It is obvious that m_t begins to perform after m_h has finished in one cycle. Another type is called noninterference, i.e., two sub-models perform in parallel without mutual interference.

As a result, we can obtain a width first search algorithm to construct a model from a model graph. The algorithm can be seen in the following graph (Figure 14).

```
//Algorithm for constructing a unified view of the integrated model from a model graph.
Input:
    C = {c1, c2, …};          //The set of connectors
    SM = {sm1, sm2,…};        //The set of sub-models
    L = {l1, l2, …};          //The set of edges
Output:
    M = ε;                    //Integrated model
Variable:
    TM = Ø;                   //The set of temporal models
  TSM = Ø;                  //The set of temporal sub-models
    tm, tm`;                  //TM's element
    tsm;                      // TSM's element
    thm;                      //Temporary head model

Begin Integration
    foreach sm∈SM
        if ∄ c ∈ C s.t. (c, sm) ∈L
            or (∃c ∈ C s.t. (c, sm) ∈L and ∄sm' ∈SM s.t. (sm', c) ∈L ){
                TM = TM ∪ {sm };    //Find out all the first group of sm
                SM = SM - {sm };
            }  //End of if

        while TM ≠Ø {
            foreach tm∈TM      //Integrating each first Parallel models group
                foreach c∈C and (tm, c) ∈L
                    foreach tm`∈TM - {tm }
                        if((tm`, c) ∈L){
                            tm = tm ↑ tm`;
                            TM = TM - {tm' };
                        }
            foreach tm∈TM{      // TM has been updated.
                TSM = Ø;
                thm = ε;
                foreach c∈C and (tm, c) ∈L
                    foreach sm∈SM and (c, sm) ∈L{
                        TSM = TSM ∪ {sm};
                        SM = SM - {sm};
                    }
                    foreach tsm∈TSM          //Find out Parallel models
                        if ∃tsm' ∈TSM and tsm'≠tsm
                            s.t. tsm'→tsm or tsm' →→tsm{
                                TSM = TSM - {sm};
                                SM = SM ∪ {sm};
                            }
                    foreach tsm∈TSM         //Integrating a group of Parallel models
                        thm = thm↑tsm;

                    tm = tm→thm;            //Integrating Sequence models
            }//End of foreach tm∈TM
        }//End of while TM ≠Ø

        foreach tm∈TM
            m =   m↑tm;                     //Integrating all Parallel models at last
End Integration
```

Figure 14. The algorithm for constructing a unified view of the integrated model from a model graph.

From the algorithm, we know that the integration of many sub-models can also be taken as a model.

6. Results

Although there are few formal frameworks for IEM being studied specially, each IEM platform has its own potential formalist basis. We compare our formal framework with several platforms or standards to illustrate our framework's characteristics (in Table 1). The selected platforms are OMS3 [39], OpenMI [8] and ESMF [40].

Table 1. The comparison with other platforms.

		Our Framework	OMS3	OpenMI	ESMF
Technological Basis		/	**Object-Oriented**		**Component-Based**
		/	**DSL + POJO + Annotation**	**Interface-Based**	**Mediator Pattern**
Formalism	**Dataflow**	yes	yes	yes	yes
	sequence	yes	yes	yes	yes
	feedback	yes	yes	yes	yes
	select	yes	no	extra codes	extra codes
	iterate	yes	no	extra codes	extra codes
Integration reusable		easy	difficult	difficult	difficult

From the comparison, we can see that there are some advantages embedded in our framework. The formal framework can be used as the semantic basis of an IEM Domain Specific Language (DSL) with which the complicated model can be integrated from pre-existing models. Based on the framework, a light weighted IEM DSL named irDSL (integration-reusable Domain Specific Language) has been developed in our recent work. In Appendix A, some code snippet with irDSL for the integrated model in Figure 1 is listed. The additional details of irDSL will be discussed in another paper.

7. Conclusions and Future Work

In this paper, a formal framework for the IEM system is proposed. In the framework, the features of the model are divided into two parts, i.e., the static and dynamic features. The static features include the traditional parameters, input variables, output variables, and functions of the model. The dynamic feature is the transformation of the static features during the simulation and is represented as an FSM, which adapts the integrated model to dynamic application scenarios. Based on the framework, a unified definition of the model is proposed that makes the integrated model more manageable and reusable, as are the simple models. The integration is also represented as a multidigraph with port. An algorithm is used to explore its sufficiency such that there is a unified representation for a multidigraph.

Our proposed definition can also be used as the interface between specification and formal verification. Based on the framework, it supports integration verification at a specified time (similar to [34,41]) such that the integration specification can be proven to be correct before its implementation; thus, an iterative cycle between the implementation and the specification can be avoided. At the same time, global understanding of the integration is increased, which makes the application easy to understand and, therefore, easy to maintain. Future work should address the enhancement of this tool by supporting the proposed methodology with additional automation features.

Acknowledgments: The authors are grateful to Professor Honggang Luo for his suggestion. We also thank J. Zhou and Peña-Haro for helpful discusses and suggestions. This work was supported by the National Natural Science Foundation of China under Grant No. 61402210 and 60973137, the Program for New Century Excellent Talents in University under Grant No. NCET-12-0250, the "Strategic Priority Research Program" of the Chinese Academy of Sciences with Grant No. XDA03030100, and the Gansu Sci.&Tech. Program under Grant No. 1104GKCA049, 1204GKCA061 and 1304GKCA018. Additional funding was supplied by The Fundamental Research Funds for the Central Universities under Grant No. lzujbky-2014-49, lzujbky-2013-k05, lzujbky-2013-43, lzujbky-2013-44

and lzujbky-2012-44, the Gansu Telecom Cuiying Research Fund under Grant No. lzudxcy-2013-4, and Google Research Awards and the Google Faculty Award, China.

Author Contributions: Gaofeng Zhang and Qingguo Zhou designed the framework. Yan Li, Chong Chen, Ri Zhou and Dan Chen tested the design.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Some Code Snippet with irDSL for the Integrated Model in Figure 1.

The following is a code snippet with irDSL for the integrated model in Figure 1. The conceptual integrated model is not bound to any concrete application and can be reused easily.

In a concrete simulation, just as in Peña-Haro's example [31], a hypothetical simplified two-dimensional-vertical flow system with a height of 20 m and a length of 200 m is used. Two different spatial discretizing schemas (such as Schema 1 with 2 zones and Schema 2 with 5 zones) are intended to be adopted, for comparison with traditional integration. Qualitatively, it can be seen that instantiation of the model needs relative few codes based on the conceptual integrated model.

```
//Conceptual integrated model named WOHYMO;
model WOHYMO{
        model WOFOST("path_WO");
        //import basic WOFOST model;
        // path_WO: the path of jar file including the POJO of WOFOST;
        model HYDRUS_1D("path_HY");   //Ditto;
        model MODFLOW("path_MO");     //Ditto;

        //A select control for WOFOST used to specify that the WOFOST
        // begins to run after the crop germinates.
        select is_WOFOST_RUN {              //Control flow
                if (!before_Emergence)
                        WOFOST;      // WOFOST doesn't run until a crop emergency
        }

        //A connector for LAI (Leaf Area Index) which is produced by WOFOST
        // and consumed by HYDRUS as its one boundary condition.
        connector LAI_WO_HY{            // the ordinary connector;
                in lai1;   out lai2;          //input and output;
                expr expr(lai2=lai1);         //indicates ordinary connector;
        }

        //simplified link for ordinary connector, implying a sequence operator;
        forward link_LAI_WO_HY{
                source WOFOST.lai;
                target HYDRUS_1Dconn1.lai;
        }

        //A connector for vBot in HYDRUS (or recharge in MODFLOW) which represents
        // the water flows from unsaturated zone to saturated zone, which is produced by
        // HYDRUS and consumed by MODFLOW as its boundary condition.
        connector vBot_HY_MO{               // the recharge from HYDRUS to MODFLOW;
                in vBot;                    //input, weak typing, will be taken as an array;
                out recharge;               //output;
                expr expr(recharge = vBot));            // has same recharge in a zone;
        }

        forward link_ vBot_HY_conn {
                source HYDRUS_1D.vBot;
                target vBot_HY_MO.vBot;
        }
```

```
        forward link_ recharge_conn_HY {
            source vBot_HY_MO.recharge;
            target   MODFLOW.recharge;
        }

        //A connector for Hb (pressure head) from MODFLOW to HYDRUS H or Hb is
        // produced by MODFLOW and transferred to HYDRUS-1D as its boundary
        // condition.
        connector Hb_MO_HY {              // the H or Hb from MODFLOW to HYDRUS;
            in H;                          //input, weak typing, ;
            out Hb;                        //output, weak typing, will be taken as an array;
            expr expr(Hb = H);
        }

        backward link_Hb_MO_conn {          //implying a feedback operator;
            source   MODFLOW.H;
            target   Hb_MO_HY.H;
        }

        forward   link_ Hb_conn_HY {
            source   Hb_MO_HY.Hb;
            target   HYDRUS_1D.Hb;
        }

        relation   m2o   <HYDRUS_1D,MODFLOW>;   //m2o relation, for extendability;
        relation   o2m   < MODFLOW, HYDRUS_1D >; //o2m relation;

        //other codes are omitted due to space limitations.
    }
    //Instance of WOHYMO with the 5 zones
    model WOHYMO WOHOMO_5 {          //instance definition;
        model WOFOST inst_WOFOST [5];          //5 instances of WOFOST;
        model HYDRUS inst_HYDRUS [5];              //5 instances of HYDRUS;
        model MODFLOW inst_MODFLOW;          //1 instances of MODFLOW;
        for i in [1..5]                          //o2o relation;
                inst_WOFOST [i] ->inst_ HYDRUS [i];
        inst_HYDRUS [1..5] ->inst_ MODFLOW;
        //5 inst_HYDRUSes relates to 1 inst_MODFLOW;
        inst_MODFLOW ->inst_HYDRUS [1..5];
        //1 inst_MODFLOW relates to 5 inst_HYDRUSes;
}

    //instance of ex16_WOHYMO with the 2 zones
    model WOHYMO WOHOMO_2 {          //instance definition;
        model WOFOST inst_WOFOST [2];          //2 instances of WOFOST;
        model HYDRUS inst_HYDRUS [2];          //2 instances of HYDRUS;
        model MODFLOW inst_MODFLOW;          //1 instances of MODFLOW;
        for i in [1..2]                          //o2o relation;
                inst_WOFOST [i] ->inst_ HYDRUS [i];
        inst_HYDRUS [1..2] ->inst_ MODFLOW;
          //2 inst_ HYDRUSes relates to 1 inst_ MODFLOW;
        inst_MODFLOW ->inst_HYDRUS [1..2];
          //1 inst_MODFLOW relates to 2 inst_HYDRUSes;
}
```

References

1. Bailey, G.W.; Mulkey, L.A.; Swank, R.R. Environmental implications of conservation tillage: A system approach. In *A System Approach to Conservation Tillage*; D'Itri, F.M., Ed.; Lewis Publishers Inc.: Chelsea, MI, USA, 1985; pp. 239–265.

2. Cohen, Y. *Pollutants in a Multimedia Environmental*; Plenum Press: New York, NY, USA, 1986.
3. Mackay, D. *Multimedia Environmental Models: The Fugacity Approach*; Lewis Publishers: Michigan, MI, USA, 1991.
4. Walters, C.J. *Adaptive Management of Renewable Resources*; Macmillan Publishing Co.: New York, NY, USA, 1986.
5. Voinov, A.; Shugart, H. 'Integronsters', integral and integrated modelling. *Environ. Model. Softw.* **2013**, *39*, 149–158. [CrossRef]
6. Laniak, G.F.; Olchin, G.; Goodall, J.; Voinov, A.; Hill, M.; Glynn, P.; Whelan, G.; Geller, G.; Quinn, N.; Blind, M.; et al. Integrated environmental modelling: a vision and roadmap for future. *Environ. Model. Softw.* **2013**, *39*, 3–23. [CrossRef]
7. Leavesley, G.H.; Markstrom, S.L.; Brewer, M.S.; Viger, R.J. The Modular Modelling System (MMS)—The physical process modelling component of a database-centered decision support system for water and power management. *Water Air Soil Pollut.* **1996**, *90*, 303–311. [CrossRef]
8. OpenMI. The OpenMI Open Midel Interface Project. 2016. Available online: https://publicwiki.deltares.nl/display/OPENMI/Version+2.0 (accessed on 24 December 2016).
9. David, O.; Markstrom, S.L.; Rojas, K.W.; Ahuja, L.R.; Schneider, W. The object modelling system. In *Agricultural System Models in Field Research and Technology Transfer*; Ahuja, L.R., Ma, L., Howell, T.A., Eds.; Lewis Publishers: Boca Raton, FL, USA, 2002; pp. 317–344.
10. David, O.; Ascough, J.C., II; Lloyd, W.; Green, T.R.; Rojas, K.W.; Leavesley, G.H.; Ahuja, L.R. A software engineering perspective on environmental modelling framework design: The object modelling system. *Environ. Model. Softw.* **2013**, *39*, 201–213. [CrossRef]
11. Hill, C.; DeLuca, C.; Balaji, V.; Suarez, M.; da Silva, A. The architecture of the earth system modelling framework. *Comput. Sci. Eng.* **2004**, *6*, 18–28. [CrossRef]
12. Thurman, D.A.; Cowell, A.J.; Taira, R.Y.; Frodge, J. Designing a collaborative problem solving environment for integrated water resource modelling. In *Brownfields: Multimedia Modelling and Assessment*; Whelan, G., Ed.; WIT Press: Southampton, UK, 2004.
13. Aquaveo. *Water Modelling Solutions*; Aquaveo: Provo, UT, USA, 2012; Available online: http://www.aquaveo.com/ (accessed on 8 December 2013).
14. Peckham, S.D. Evaluation of model coupling frameworks for use by the Community Surface Dynamics Modelling System (CSDMS). In Proceedings of the MODFLOW and MORE, Golden, CO, USA, 18 May 2008.
15. van Ittersum, M.K.; Ewert, F.; Heckelei, T.; Wery, J.; Olsson, J.A.; Andersen, E.; Bezlepkina, I.; Brouwer, F.; Donatelli, M.; Flichman, G.; et al. Integrated assessment of agricultural systems e a component-based framework for the European Union (SEAMLESS). *Agric. Syst.* **2008**, *96*, 150–165. [CrossRef]
16. Parker, D.; Manson, S.; Janssen, M.; Hoffmann, M.; Deadman, P. Multi-agent systems for the simulation of land-use and land-cover change: A review. *Ann. Assoc. Am. Geogr.* **2003**, *93*, 314–337. [CrossRef]
17. Zhao, J.; Cai, X.; Wang, Z. Comparing administered and market-based water allocation systems through a consistent agent-based modelling framework. *J. Environ. Manag.* **2013**, *123*, 120–130. [CrossRef] [PubMed]
18. Zhao, G.; Bryan, B.A.; King, D.; Luo, Z.; Wang, E.; Bende-Michlc, U.; Song, X.; Yu, Q. Largescale, high-resolution agricultural systems modelling using a hybrid approach combining grid computing and parallel processing. *Environ. Model. Softw.* **2013**, *41*, 231–238. [CrossRef]
19. Yalew, S.; van Griensven, A.; Ray, N.; Kokoszkiewicz, L.; Betrie, G.D. Distributed computation of large scale SWAT models on the Grid. *Environ. Model. Softw.* **2013**, *41*, 223–230. [CrossRef]
20. Granell, C.; Díaz, L.; Gould, M. Service-oriented applications for environmental models: Reusable geospatial services. *Environ. Model. Softw.* **2010**, *25*, 182–198. [CrossRef]
21. Goodall, J.L.; Robinson, B.F.; Castronova, A.M. Modelling water resource systems using a service-oriented computing paradigm. *Environ. Model. Softw.* **2011**, *26*, 573–582. [CrossRef]
22. Bastin, L.; Cornford, D.; Jones, R.; Heuvelink, G.B.M.; Pebesma, E.; Stasch, C.; Nativi, S.; Mazzetti, P.; Williams, M. Managing uncertainty in integrated environmental modelling: The UncertWeb framework. *Environ. Model. Softw.* **2013**, *39*, 116–134. [CrossRef]
23. Wing, J.M. A specifier's introduction to formal methods. *Computer* **1990**, *23*, 8–24. [CrossRef]
24. Argent, R.M. An overview of model integration for environmental applications-components, frameworks and semantics. *Environ. Model. Softw.* **2004**, *19*, 219–234. [CrossRef]

25. Argent, R.M.; Voinov, A.; Maxwell, T.; Cuddy, S.M.; Rahman, J.M.; Seaton, S.; Vertessy, R.A.; Braddock, R.D. Comparing modelling frameworks: A workshop approach. *Environ. Model. Softw.* **2006**, *21*, 895–910. [CrossRef]
26. Voinov, A.; Cerco, C. Model integration and the role of data. *Environ. Model. Softw.* **2006**, *25*, 965–969. [CrossRef]
27. Rizzoli, A.E.; Donatelli, M.; Athanasiadis, I.N.; Villa, F.; Huber, D. Semantic links in integrated modelling frameworks. *Math. Comput. Simul.* **2007**, *78*, 412–423. [CrossRef]
28. Schmitz, O.; Karssebnerg, D.; de Jong, K.; de Kok, J.-L. Constructing integrated models: A scheduler to execute coupled components. In Proceedings of the AGILE 2011, the 14th AGILE International Conference on Geographic Information Science, Advancing Geoinformation Science for a Changing World, Utrecht, The Netherlands, 18 April 2011.
29. Kragt, M.E.; Robson, B.J.; Macleod, C.J.A. Modellers roles in Structuring integrative research projects. *Environ. Model. Softw.* **2013**, *39*, 322–330. [CrossRef]
30. Lloyd, W.; David, O.; Ascough, J.C., II; Rojas, K.W.; Carlson, J.R.; Leavesley, G.H.; Krause, P.; Green, T.R.; Ahuja, L.R. Environmental modelling framework invasiveness: Analysis and implications. *Environ. Model. Softw.* **2011**, *26*, 1240–1250. [CrossRef]
31. Peña-Haro, S.; Zhou, J.; Zhang, G.; Chen, C.; Stauffer, F.; Kinzelbach, W. A multi-approach framework to couple independent models for simulating the interaction between crop growth and unsaturated-saturated flow processes. In Proceedings of the International Environmental Modelling and Software Society (iEMSs) 2012 International Congress on Environmental Modelling and Software: Managing Resources of a Limited Planet: Pathways and Visions under Uncertainty, Sixth Biennial Meeting (iEMSs 2012), Leipzig, Germany, 1 July 2012; pp. 1224–1231.
32. Zhang, G.; Zhou, J.; Zhou, Q.; Cheng, G.; Li, X. Integrated Eco-hydrological modelling by a combination of coupled-model and algorithm using OMS3. In Proceedings of the International Environmental Modelling and Software Society (iEMSs) 2012 International Congress on Environmental Modelling and Software: Managing Resources of a Limited Planet: Pathways and Visions under Uncertainty, Sixth Biennial Meeting (iEMSs 2012), Leipzig, Germany, 1 July 2012; pp. 1201–1207.
33. Hamadi, R.; Benatallah, B. A Petri net-based model for web service composition. In Proceedings of the ADC '03 14th Australasian database conference of CRPIT, Adelaide, Australia, 1 February 2003; Volume 17.
34. Dumez, C.; Bakhouya, M.; Gaber, J.; Wack, M.; Lorenz, P. Model-driven approach supporting formal verification for web service composition protocols. *J. Netw. Comput. Appl.* **2013**, *36*, 1102–1115. [CrossRef]
35. Lomazova, I. Nested Petri nets—A formalism for specification and verification of multi-agent distributed systems. *Fundam. Inf.* **2000**, *43*, 195–214.
36. Alur, R.; Yannakakis, M. Model checking of hierarchical state machines. In Proceedings of the Sixth ACM FSE, Orlando, FL, USA, 1 November 1998.
37. Girault, A.; Lee, B.; Lee, E.A. Hierarchical finite state machines with multiple concurrency models. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.* **1999**, *18*, 742–760. [CrossRef]
38. Lee, E.A.; Tripakis, S. Modal models in Ptolemy. In Proceedings of the 3rd International Workshop on Equation-Based Object-Oriented Modelling Languages and Tools (EOOLT), Oslo, Norway, 3 October 2010; Volume 47, pp. 11–21.
39. OMS3. The OMS3 Doc Website. 2011. Available online: http://nrrc.ars.usda.gov/ModelFrameworks/ObjectModelingSystem/Documentation.aspx (accessed on 20 September 2016).
40. ESMF. The ESMF User Doc. 2014. Available online: http://www.earthsystemmodeling.org/esmf_releases/public/last/ESMF_usrdoc/ESMF_usrdoc.html (accessed on 20 January 2017).
41. Dustdar, S.; Zdun, U. Model-driven and pattern-based integration of process-driven SOA Models. *Int. J. Bus. Process Integr. Manag.* **2006**, *2*, 109–119.