Simulation

Simulation: Transactions of the Society for Modeling and Simulation International
2016, Vol. 92(8) 747–769
© The Author(s) 2016
DOI: 10.1177/0037549716657168
sim.sagepub.com

System entity structure extension to integrate abstraction hierarchies and time granularity into DEVS modeling and simulation

Jean-François Santucci[1*], Laurent Capocchi[1*] and Bernard P Zeigler[2,3*]

Abstract
The modeling and simulation (M&S) of complex systems often requires models described at different levels of detail characterized by differences in abstraction hierarchies and/or time granularity. The discrete-event system specification (DEVS) is a framework based on mathematical systems theory that offers a computational basis for application of M&S to systems engineering and that has become widely adopted for its support of discrete-event, continuous, and hybrid applications. A fundamental representation of DEVS hierarchical modular model structures is the system entity structure (SES), which represents a design space via the elements of a system and their relationships in a hierarchical and axiomatic manner. As has been described in a number of publications, the SES supports development, pruning, and generation of DEVS simulation models. The goal of this paper is to propose an extension of SES in order to integrate both the concepts of abstraction hierarchies and time granularity into DEVS. This paper explains in detail: (i) the concepts of abstraction hierarchies and time granularity; (ii) the extension of SES in order to take into account these concepts; (iii) DEVS M&S of complex systems according to different levels of detail (abstraction hierarchies and time granularity); (iv) the use of a Python DEVS simulator (DEVSimPy) to implement the management of abstraction hierarchies and time granularity. A real case study is given to illustrate the proposed approach, and follow-on research needed to implement the concepts is discussed.

Keywords
Discrete-event formalism, abstraction hierarchy, time granularity, system entity structure, DEVS, DEVSimPy

1. Introduction

Systems engineering can be viewed as a strategy for the design of systems where the search space of alternative designs is practically infinite. A key element in executing such a strategy is the use of abstraction to simplify the alternative structures, enabling tractable generation and testing of smaller combinatorial spaces. Here, abstractions, or, more concretely, models based on them, are derived by temporarily ignoring overall system objectives, requirements, or constraints in favor of working with more-easily satisfied formulations of these criteria. Modeling and simulation (M&S) is an important methodology for system engineering that supports such abstraction-based search.[1] However, it often focuses on the back-end generation and testing of system models, once developed, for achievement of (sub-optimal) requirements, rather than on the front-end development and management of the abstractions underlying such models. However, isolated consideration from simplified perspectives is not sufficient for achievement of today's complex systems' robust design requirements. This mandates more effort towards a fully-fledged search through the infinite space of alternatives. In this paper, we seek to move towards a new theory for expression, development, management, and realization of abstractions in M&S to support systems engineering as a search strategy through infinite design spaces.

[1]Department of Computer Science, University of Corsica, France
[2]Arizona Center for Integrative Modeling and Simulation, University of Arizona, USA
[3]RTSync Corporation, MD, USA
[*]SCS member.

Corresponding author:
Jean-François Santucci, University of Corsica, Campus Grimaldi, 20250, Corte, France.
Email: santucci@univ-corse.fr

Our approach is based on working with artifacts of expression in M&S associated with the discrete-event system specification (DEVS) formalism.[2,3] This is a framework based on mathematical systems theory that offers a computational basis for application of M&S to systems engineering and that has become widely adopted for its support of discrete-event, continuous, and hybrid applications. A fundamental representation of DEVS hierarchical modular model structures is the system entity structure (SES), which represents a design space via the elements of a system and their relationships in hierarchical and axiomatic manner. SES is a declarative knowledge representation scheme that characterizes the structure of a family of models in terms of decompositions, component taxonomies, and coupling specifications and constraints.[4] As has been described in a number of publications[1,5–8] SES supports development, pruning, and generation of a family of DEVS simulation models.

One pillar of this research is to propose an extension of SES in order to integrate the concepts of abstraction hierarchies and time granularity into DEVS. To accomplish this task, we detail the set of actions that have to be performed based on the previous work[9]: (i) formally define abstraction hierarchies and time granularity; (ii) add new abstraction hierarchy and time granularity aspects to SES; and (iii) implement abstraction hierarchies and time granularity into DEVS M&S based on the newly introduced SES aspects. When dealing with M&S of complex systems, the first step concerns the definition of models. The development of models depends on the knowledge of the designer, the domain to which it belongs, and the perspective for which the modeling phase is accomplished. This perspective is actually a mechanism that allows the designer to determine the required relevant information.

The concept of abstraction hierarchies makes it possible to consider information that is relevant by considering abstractions as a means of selectively hiding elements that do not reveal what is essential to objectives.[10] It can be noted that the amount of information decreases as the level increases; a model described at a low level of abstraction contains more information than a model at a higher a level of abstraction. ‘‘Determining the correct abstraction level refers to selecting the quantum of information that must be included in the model to help address the modeling goals.’’[10] Therefore, properly defining the abstraction level is an important step in modeling and is often done very early in the modeling process. A model that contains different levels of abstraction is called a ‘‘hierarchical model.’’[11] A model described according to several abstraction levels is designed according to an ‘‘abstraction hierarchy’’ scheme. This stratification is intended for gradual inclusion of details as required by more-incisive objectives and system design requirements. Thus, this hierarchy allows for a given level of abstraction to focus on key system-related data, and to relegate irrelevant information to other levels of abstraction.

The concept of time granularity is as important as the abstraction hierarchy in model construction. In many cases, the systems we are trying to model and simulate contain elements that are not defined on the same time base. In order to take into account the disparity between the elements of a system, we propose the definition of a temporal hierarchy. This hierarchy provides the opportunity to represent the level of temporal details (more or less fine). This approach is defined as a contraction or expansion of space–time. We obtain a series of time-scales for change, depending on the selected temporal level, to consider only the relevant information, in order to obtain efficient results. Thus, an event defined at a given temporal level (e.g., the hour) is transformed into a set of events at a finer temporal level (e.g., the second). This approach is justified by the fact that the model builder does not always have the information at the same time basis.

The second step involved in M&S of complex systems concerns the simulation part. In the case of models described at different levels of abstraction, the simulation part has to take into account the transfer of information when going from one level to another. In the same way, in the case of models described according to several levels of time granularity, the simulation part has to take into account the conversion of event messages exchanges when going to a finer or coarser level of time granularity. Mechanisms allowing the propagation of events through models described according to several levels of abstraction or several levels of time granularity have to be carefully defined.

The introduction of abstraction hierarchies and time granularity into both SES and DEVS formalism will be useful when trying to tackle the following problems: multi-abstraction modeling, approximate DEVS simulations for increased performance, approximate M&S for lower model development time, and interoperability between models at different levels of abstraction. We outline the three most important components of the challenge tackled in this paper.

- SES extension challenge: SES is used to describe how a system is decomposed into sub-systems allowing the representation of different specializations of a system that might occur. However, integrating both abstraction and time specializations into SES is still a task that has to be accomplished. We facilitate addressing this challenge by providing a framework to deal with both abstraction and time hierarchy that is based on making it possible for the user to specify different levels of abstraction or time hierarchies and how they are communicated between each other.
- Abstraction hierarchy challenge: Being able to deal with several levels of abstraction when using a classical or parallel DEVS (PDEVS) modeling and an

associated simulation scheme is an important challenge for the DEVS community. This is becoming an increasingly significant challenge as systems to be modeled and simulated using the DEVS formalism are becoming more and more complex. In this paper we seek a way to integrate abstraction hierarchies into DEVS and allow a user to simulate models described at different levels of abstraction using the DEVS or PDEVS abstract simulator.

- Time granularity challenge: The issue of time granularity occurs when there is mismatch between two interconnected atomic models regarding how often the information is generated and consumed. For example, a weather simulation model may be required to couple models driving weather information on an hourly or daily basis. This mismatch of time granularity makes it ineffective to directly couple these kinds of models, which are at two different levels of time granularity. In this paper, we show how to construct and manage time hierarchy into DEVS and allow a user to simulate models described at different levels of time granularity.

It is important to highlight that integrating the concept of abstraction hierarchies and time granularity into DEVS allows the management of levels of hierarchies into DEVS without any involvement of SES. Furthermore, it is important to point out that the proposed approach fits within both classic and parallel DEVS formalism.

The rest of the paper is organized as follows. The following section presents the context of the work. Specifically, the notions of abstraction hierarchies and time granularity are presented. The SES as well as the DEVS formalism and the DEVSimPy framework are also introduced. Section 3 first provides related work and the main contribution of the paper. Then, it deals with the integration of abstraction hierarchies and time granularity in the framework of both the SES and the DEVS formalism. The implementation of the proposed approach is presented in Section 4. The notions of levels of abstraction and levels of time granularity are introduced in the DEVSimPy software framework. We also describe a real case study dealing with behavior of a watershed, which is used in order to validate the implementation. The SES and the DEVSimPy M&S results are presented in detail. Section 5 concludes and proposes future work.

2. Background

2.1. DEVS formalism and DEVSimPy framework

The DEVS formalism was introduced by Zeigler in the seventies[2] for modeling discrete-event systems in a hierarchical and modular way. DEVS formalizes what a model is, what it must contain, and what it does not contain (experimentation and simulation control parameters are not contained in the model). Moreover, DEVS is universal and unique for discrete-event system models. Any system that accepts events as inputs over time and generates events as outputs over time is equivalent to a DEVS. DEVS allows automatic simulation on multiple different execution platforms, including those on desktops (for development) and those on high-performance platforms (such as multi-core processors). With DEVS, a model of a large system can be decomposed into smaller component models with couplings between them. DEVS formalism defines two kinds of models: (i) atomic models that represent the basic models, providing specifications for the dynamics of a sub-system using function transitions; (ii) coupled models that describe how to couple several component models (which can be atomic or coupled models) together to form a new model. This hierarchy, inherent to the DEVS formalism, can be called a ‘‘description hierarchy’’ by allowing the definition of a model using a hierarchical decomposition. It should be pointed out that this kind of hierarchy does not involve any abstraction level definition since the behaviors of all implied models are defined at the same level of abstraction. However, as a hierarchy of description, the hierarchical decomposition in DEVS may still be regarded a kind of abstraction: in top-down design, modelers are able to consider couplings and interfaces between models without considering details of internal components.

An atomic DEVS model can be considered as an automaton with a set of states and transition functions allowing the state change when an event occurs or not. When no events occurs, the state of the atomic model can be changed by an internal transition function noted δ_{int}. When an external event occurs, the atomic model can intercept it and change its state by applying an external transition function noted δ_{ext}. The lifetime of a state is determined by a time advance function called t_a. Each state change can produce output message via an output function called λ. A simulator is associated with the DEVS formalism in order to exercise instructions of coupled model to actually generate its behavior. The architecture of a DEVS simulation system is derived from the abstract simulator concepts associated with the hierarchical and modular DEVS formalism. The PDEVS[12] extends the classic DEVS essentially by allowing bags of inputs to the external transition function. Bags can collect inputs that are built at the same date, and process their effects on the outputs, which will result in new bags. This formalism offers a solution to managing simultaneous events that could not be easily managed with classic DEVS.

DEVSimPy (https://github.com/capocchi/DEVSimPy) (Python Simulator for DEVS models)[13] is a user-friendly interface for collaborative M&S of DEVS systems implemented in the Python (http://python.org) language. DEVSimPy is an open source project available under the

GPL V3 license and its development is supported by the University of Corsica ''Pasquale Paoli'' Computer Science research team. The DEVSimPy project uses the Python programing language for providing a graphical user interface (GUI) (based on the wxPython (http://www.wxpython.org) graphic library) for the PyDEVS and PyPDEVS (http://msdl.cs.mcgill.ca/projects/DEVS/PythonPDEVS,[14] the PDEVS implementation of PyDEVS) application programming interfaces (APIs). DEVSimPy has been set up to facilitate both the coupling and the reusability of the PyDEVS classic DEVS models and the PyPDEVS Python PDEVS models. Moreover, the DEVSimPy architecture is based on an model-view-controller (MVC) pattern coupled with the aspect-oriented programing concept, which renders the user interface and the simulation kernel (PyDEVS or PyPDEVS) independent. Users can select the desired simulation kernel for DEVSimPy models that are compatible with (a wrapper is provided for this) the Py(P)DEVS simulators. In this way, when the simulators' code sources are updated, the DEVSimPy GUI part is not affected. It should be noted that PyDEVS and PyPDEVS are also used in the excellent multi-modeling GUI software named ATOM3, and its excellent and promising successor ATOMPM.[15]

With DEVSimPy, models can be stored in a library in order to be reused and shared (1 in Figure 1). More specifically, a DEVSimPy model is a compressed file composed by a Python file (behavioral specifications according to Py(P)DEVS specifications) and a text file (graphical view according to wxWidgets API). When a model is instantiated, the corresponding compressed file is extracted and: (i) the graphical representation is constructed from the text file (ii) and the behavior is instantiated from the Python file. Thus, the view and the behavior of a model are split, and a user can change a behavior of a model (by changing the Python file) without changing its graphical view. The same strategic storing approach was taken in Fard and Sarjoughian[16] where the data for every atomic model is stored in two flat files (''domain'' and ''diagram'' files). A set of DEVSimPy models constitutes a shared library due to the fact that all models can be loaded or updated from an external location such as a file server (Dropbox, GoogleDrive, GitHub, etc.), which could be also considered as a kind of ''online model store''. However, it should be stated that this concept of model store can be significantly extended with the concept of model repositories, as discussed in Chapter 16 of Zeigler and Sarjoughian[1] or in Sarjoughian and Elamvazhuthi,[17] where CoSMoS stores the behavioral model (''domain'' file) with its corresponding structural model in a database repository. Then, it generates visual representations of the models using a set of rules.

Nevertheless, a Python file with Py(P)DEVS specifications is embedded by DEVSimPy, which transforms it into an object with a default graphical view when it is dropped in the interface (2 in Figure 1). The creation of dynamic libraries composed with DEVS components is easy, since the user is coached by dialogs and wizards during the building process. With DEVSimPy, complex systems can be modeled by a coupling of DEVS models (2 in Figure 1) and the simulation is performed in an automatic way. Moreover, DEVSimPy allows the extension (or the override) of their features in using special plug-ins managed in a modular way.[18]

In order to implement the abstraction hierarchy and time granularity concepts proposed in this paper, a new branch has been developed from the master branch of the

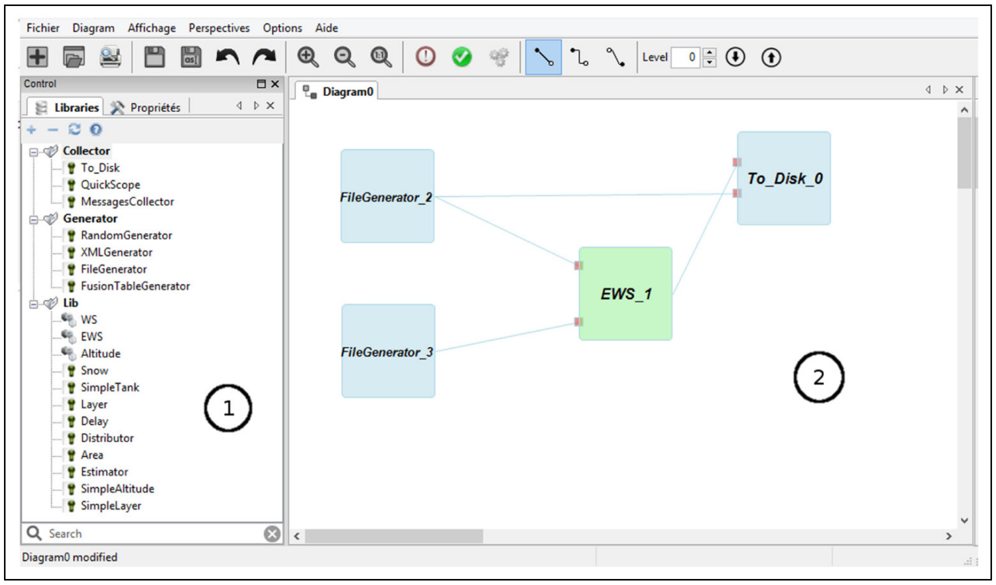

Figure 1. DEVSimPy general graphical interface.

DEVSimPy GitHub repository (https://github.com/capocchi/DEVSimPy/tree/Hierarchy). From the modeling point of view, a model that embeds the abstraction hierarchy or the time granularity encapsulates new files (since it is a compressed file in a dynamic library) that isolate this information from the rest of the model specifications (behavior and view). Concerning the interoperability of models with different time granularities, and at different levels of abstraction, it is possible to include models belonging to different libraries. The composition and the level of abstraction/granularity is defined by the user during the instantiation of the models. If the user wants to implement a new level of detail (abstraction or time), it will be shared with other users once it is saved in the dynamic library.

2.2. What are abstraction hierarchies?

The purpose of abstraction hierarchies is to hide information and manage complexity. To be useful, individuals must be able to work independently at each level of the hierarchy. Abstraction hierarchies are a human invention designed to assist people in engineering very complex systems by ignoring unnecessary details. If, instead, an abstraction hierarchy is specified, it allows the designer of a complex system to ignore some of the implementation details and focus only on the high-level design issues. Engineers in all disciplines take advantage of abstraction hierarchies to design and build complicated systems. One of the most difficult tasks in the field of M&S of complex systems is to choose a good level of detail. In all domains, models are built at a precise abstraction level. The abstraction level of a model determines the amount of information, which is contained in the model.

We have to point out that this concept of abstraction hierarchies is quite different from the hierarchy of description inherent to the DEVS formalism. The notion of DEVS coupled models involving DEVS atomic or coupled models allows the definition of models using a hierarchy of description. Such a notion of hierarchy is only a means to easily define models. Each level of description is independent (there is no relation between levels) and the corresponding coupled model interface (input/output ports configuration) is unchanged. In that case, a flattened model can be easily generated from this hierarchy of description. In contrast, it is not as easy to derive a flattened model when dealing with models involved in abstraction hierarchies, since data transfers have to be performed.

The left-hand side of Figure 2 illustrates the concept of the hierarchy of abstraction. It highlights how a model *Node N* can be defined at several different levels of abstraction. At level N, this model has two input ports 1 and 2. At level N + 1, these two input ports are duplicated into four ports 1.1, 1.2, 2.1 and 2.2. A typical example of this concept is the transition from byte to bit in the application field of microelectronics. Obviously the information managed by nodes Na1, Na2 and Na3 at level N + 1 is more precise than the information managed by Node N at level N.

We have to explicitly highlight the difference between the concept of abstraction hierarchies and the concept of hierarchy of description (found in the DEVS formalism when dealing with coupled models). The right-hand side of Figure 2 describes the concept of the hierarchy of description, with each level further decomposing one or more components at the level below it. We note that, in contrast to the abstraction hierarchy, the interface (inputs ports) is the same for a component at each level of its description. The main differences between the two kinds of hierarchies concerns the fact that, in the hierarchy of description, the same information is expressed independently of the level of hierarchy. In contrast, for the hierarchy of abstractions, going from one level of hierarchy to another does imply an increase in the information (addition of detail), which has to be transferred (disaggregated) from ports at level N to corresponding ports at level N + 1

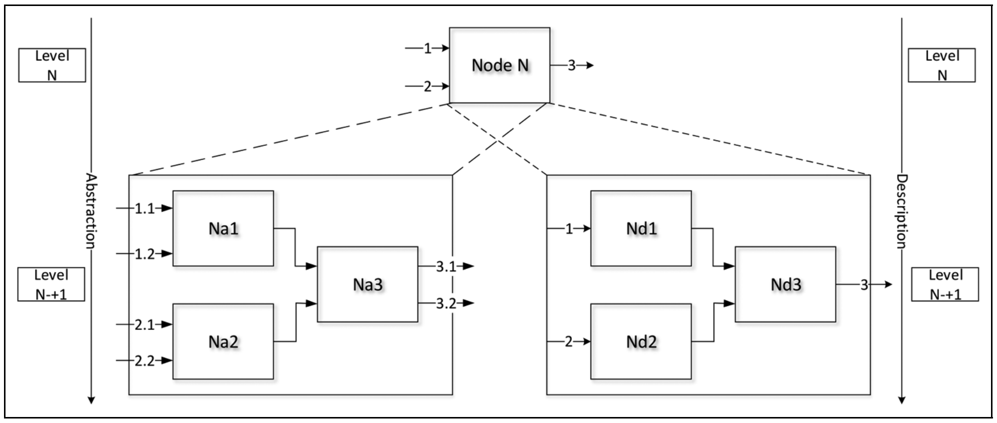

Figure 2. Hierarchy of description and abstraction for the model "Node N".

(the interface is modified). Conversely, the information at the output ports of components at level N + 1 must be reduced (aggregated) to appear at corresponding ports at level N. A way of looking at this is that, in the description hierarchy, flattening the model (removing its hierarchical structure)[2] will not change its behavior – showing that all information is ultimately in the atomic models and their interactions. In contrast, flattening is not appropriate for abstraction hierarchies since different information (in abstracted form) resides at each level.

2.3. *What is time granularity?*

Another kind of hierarchy concerns the temporal aspect. A temporal position can be described at various levels of granularity of time depending on the required accuracy or the available system knowledge. This not only means that we can use different units of time when performing M&S to represent and manipulate quantities of time in a given model, but it involves moving from one level to a finer/coarser level of time granularity. For example, a modeler can describe a system considering days as the unit of time, but might need for some specific parts of the model to switch to a finer time granularity, such as seconds, and for some other parts to switch to a coarser level, such as weeks. Such a capability to provide temporal representations on different ''levels grains'' is a major requirement for many simulation applications. We can mention a set of temporal granularity approaches proposed in the literature: the set theory developed by Bettini et al.[19] and the logical approach systematically studied by Montanari et al.[20,21] for quantitative time, Euzenat[22] for qualitative time granularity. To represent time in several levels of detail, a temporal granular representation is proposed in Euzenat. Euzenat[22] Such a representation manages temporal entities using different hierarchically organized spaces (called ''granularities''). The management of such entities requires the definition of conversion operators between two granularity levels in order to use the same temporal entity under different granularities.

Figure 3 gives an example of such granularities. The ''Week'' granularity is a coarser level relative to the ''Day'' granularity since ''Day'' groups into ''Week'' while ''Week'' partitions into ''Day''. This means that two conversion operators are required in order to manage such granularity: the ''Week'' granularity can be generated by applying a group conversion to the granularity ''Day'', while the ''Day'' granularity can be generated by applying a partition conversion operator to the granularity ''Week''.

In this paper we use the concept of time granularity[23] in order to model time information with respect to different temporal domains. This means that one can use different time units, e.g., days and weeks, to represent time quantities with different temporal domains of a layered temporal model and of switching from one domain to a coarser/finer one. For example, in order to specify the temporal evolution of a dam, the hydrologic experts usually use days: ''During rainy weeks, the level of the dam increases by 1 meter a day''. The description of the control device's behavior may use microseconds: ''When an alarm comes from the level sensors, send an acknowledgment signal in 50 microseconds''. Systems of such a type have different time granularities, and it is unnatural and sometimes impossible to obtain a modeling scheme of these systems by using a unique time granularity in order to describe the behavior of all the components. Obviously, there are situations in which the event should not be aggregated even though the time granularity is different. For example, considering the rise and fall of the sun in each day, there would be two events whatever the granularity level (hour, second). In this case, it is an event-to-event mapping and time granularity is not required.

2.4. *The SES ontology framework*

SES[24,25] is a formal ontology framework, axiomatically defined, to represent the elements of a system (or world) and their relationships in a hierarchical manner, making a family of hierarchical DEVS models.

Figure 4 provides a quick overview of the elements and relationships involved in a SES. *Entities* represent things that have existence in a certain domain. They can have *variables*, which can be assigned a value within given range and types. An *aspect* expresses a way of decomposing an object into more-detailed parts and is a labeled decomposition relation between the parent and the children. *Multi-aspects* are aspects for which the components are all of the one kind. *Specializations* represent

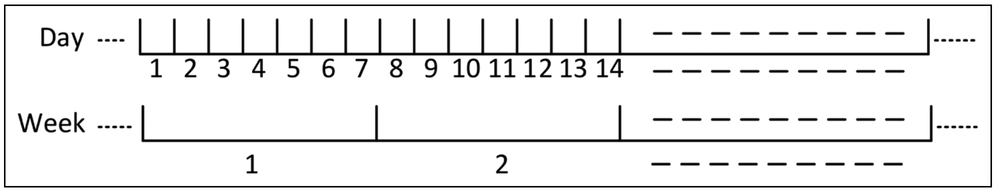

Figure 3. Day-to-week and week-to-day time granularity example.

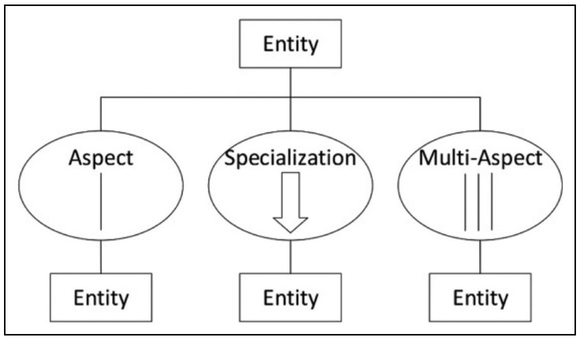

Figure 4. Overview of system entity structure (SES) items and relationships.

categorizes or families of specific forms that a thing can assume. It is a labeled relation that expresses alternative choices that a system entity can take on.

The concept of SES is illustrated with an example where we consider the way a book is constructed from physical pieces (*frontcover*, *backcover* and *pages* in Figure 5) and content (*preface* and *mainbody* in Figure 5). An aspect denotes the relationship between the object and the parts into which it has been broken. Thus, we can label the aspect representing the physical construction (respectively content construction) of a book, *physicalDec* (respectively *contentDec*). Figure 5 depicts the two items, *physicalDec* and *contentDec*, as aspects of the entity *book*. In terms of natural language, we can write: `From the physical perspective, a book is made of a front cover, pages` **and** `a back cover ; from the content perspective, a book is made of a preface` **and** `a main body`. In addition, it is important to note that an *aspect* is used when you want to represent sub-things of a thing – where "and" denotes the necessity that all of the sub-things must appear together to comprise the thing.

A specialization denotes the relationship between a general object and its variants belonging to a given category. In Figure 5 the *colorSpec* (respectively *materialSpec*) specialization denotes the colors (respectively materials) that a back cover can take on. In the restricted natural language we can write: `A back cover can be red` **or** `black in color; a back cover can be cardboard` **or** `paper in material`. It is important to note that a *Specialization* is used when you want to represent an "or" connective among sub-string of a thing – where the "or" denotes the fact that a choice of one of the variants can replace the original.

Concerning multi-aspects, we will consider the core of the book to consist of a collection of pages. In natural language, this can be expressed as: From the physical perspective, pages are made up of more than one page. The multi-aspect, *physicalMultiAsp*, captures the relationship between the entity, namely, *pages*, and its constituents, which are all instances of the same entity, namely, *page*. It is important to note that a *Multi-Aspect* is used for the same objective as an aspect except that the components are all from the same class.

A SES specifies a family of hierarchical, modular simulation models, each of which corresponds to a complete pruning of the SES. Pruning of the SES selects a particular member of the family of models to be synthesized. The mapping from SES to DEVS formalism is depicted in Figure 6, and the transformation rules are summarized in Table 1.

A SES entity can be an atomic or coupled DEVS model. If an entity of an aspect itself has an aspect, this leads to the transformation of the corresponding component into a coupled model specified by the second aspect (due to

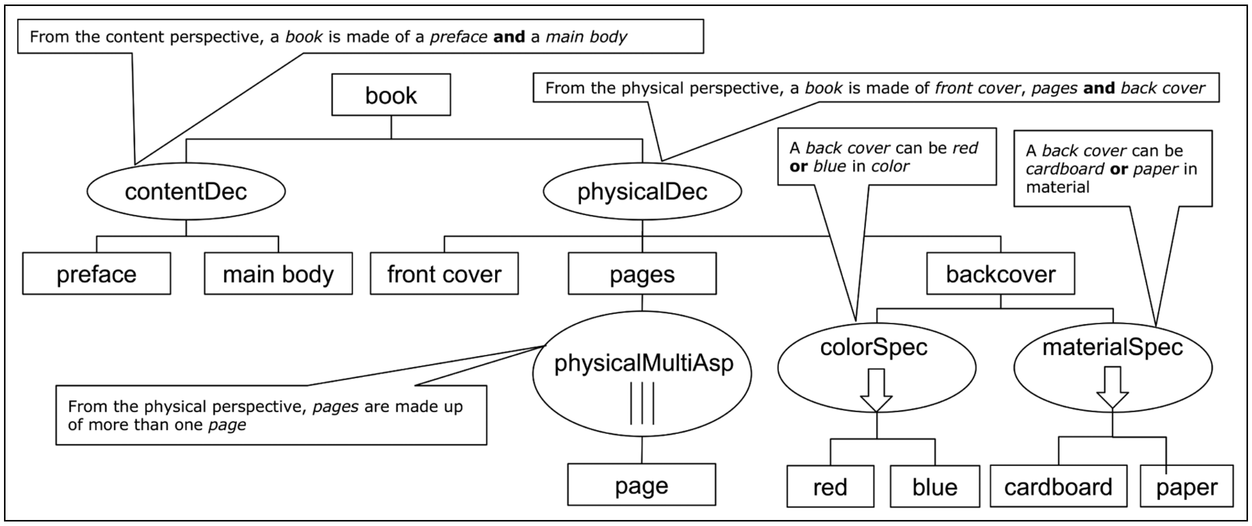
From the content perspective, a book is made of a preface and a main body
book
From the physical perspective, a book is made of front cover, pages and back cover
A back cover can be red or blue in color
A back cover can be cardboard or paper in material
contentDec
physicalDec
preface
main body
front cover
pages
backcover
physicalMultiAsp
colorSpec
materialSpec
From the physical perspective, pages are made up of more than one page
page
red
blue
cardboard
paper

Figure 5. System entity structure (SES) representation of a book.

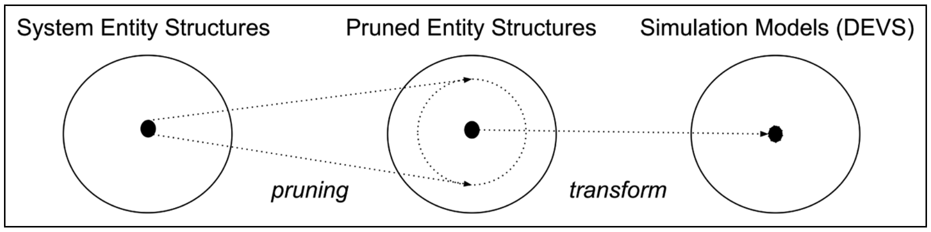

Figure 6. System entity structure (SES) to discrete event system specification (DEVS) models transformation (from Zeigler and Hammonds[24]).

Table 1. Mapping system entity structure (SES) elements to discrete event system specification (DEVS) models.

SES element Entity	DEVS model Atomic or coupled DEVS model
Aspect	Decomposition of a coupled model into DEVS models corresponding to the aspect's children. Specifies the connexion of input and output ports for information flow among the atomic/coupled models corresponding to the aspect's children
multiAspect	Decomposition of a coupled model into corresponding DEVS models, each of which is derived from the aspect's single children entity
Specialization	A family of alternative "plug-ins" for a DEVS model corresponding to the parent entity
Variables	Variables, including state variables and parameters, of the DEVS model to the entity

closure under coupling[3] of DEVS). Aspect represents the decomposition rules and the coupling between corresponding to the DEVS models children. Concerning the *multiAspect*, the mapping is similar to that for an aspect except that all models correspond to the entities generated by the *multiAspect*'s generating entity. Specialization provides a family of alternatives for a DEVS model corresponding to the parent entity. If one entity has more than one aspect, their offer alternative decomposition that can be employed to construct coupled DEVS model corresponding to the parent entity. Thus, a DEVS coupled model is made by pruning a SES, choosing one aspect and one specialization for each entity.

3. Abstraction and time hierarchies Into SES and DEVS

In this paper we show how to extend SES with both abstraction and time hierarchies; we also define how to simulate DEVS models involving several levels of abstraction as well as several levels of time granularities. Even if very little work deals with SES extension towards abstraction or time hierarchies, we can find in the literature a set of work belonging to the two last areas of research: how to manage abstraction hierarchies in the DEVS formalism, and how to deal with time granularities using DEVS. The present section provides an overview, and we propose a comparison of our approach with existing work.

3.1. Related work

The M&S of complex systems with multiple levels of abstraction was introduced several years ago.[3,11] Gradually over the years, and as indicated in Tekinay et al.,[26] research activities have been carried out to study and prove the importance of developing and maintaining multi-level models for complex systems. Concerning the DEVS formalism, the ML-DEVS (multi-level discrete-event system specification) has been introduced as a variant of the DEVS formalism[27] in order to deal with variable (dynamic) structure models. Initially envisaged for biological systems, the ML-DEVS formalism is based on macro- and micro- DEVS models used in order to extend the atomic and coupled original DEVS models. The main difference between the scheme proposed by ML-DEVS and DEVS is that macro models (coupled model for DEVS) have a state, but also and above all, a behavior and special functions (upward/downward) that fit the behavior of the macro level with the behavior at the micro level. Furthermore, the abstract simulator of ML-DEVS is close to a simplified version of the PDEVS simulator, and it has been modified from the original abstract DEVS simulator. In terms of implementation, ML-DEVS has been proposed in the general and open framework JAMES II[28] and it is often used for agent modeling. In Mittal,[29] the author presents the origin of the ML-DEVS formalism and also details the evolution of DEVS formalism. The author demonstrates that "Micro-DEVS is unsuitable for modeling complex adaptive system (CAS)'' due to the fact that an agent can be ''proactive and adaptive with learning behavior''. However, it proposes to augment the ML-DEVS formalism with detailed propositions. The multi-perspective DEVS modeling proposed in Seck and Honig[30] gives a solution for modeling complex systems according to several levels of abstraction using the specification of two DEVS models, called ''aspect'' and ''bridge'' models, in order to bring a solution to the problem of the management of alternative system perspectives as well as the identification of the links between them. Another abstraction-based modeling approach is multi-resolution modeling (MRM). MRM is a modeling technology for complex systems that considers them as hierarchical models having different resolutions at different abstraction levels.[31] As already mentioned in Moon and Hong[32] for the M&S domain, ''the level of abstraction of a model determines the amount of information contained in the model''. The resolution is defined in Moon and Hong[32] as ''the degree of detail and precision in describing real-world aspects in a model''. However, according to Davis and Bigelow,[31] abstraction and resolution can be interchangeable. MRM is an important technique used in software development research for modularity and reuse, and it is used also in the discrete-event M&S field. Again concerning DEVS formalism, Baohong and Kedi[33] propose to use the dynamic structure DEVS (DSDEVS)[34] formalism in order to manage the resolution between levels.

Concerning time granularities in DEVS, we can mention three approaches.[23,30,35] In Guo et al.[23] the time granularity is performed in the framework of simulation service composition using the DEVS formalism by introducing a time granularity handling with a component that acts as a mediator between two services. In Seck and Honig,[30] the authors deal with abstraction hierarchies. However, they mention time hierarchies by saying that further research will need to define some extension of the formalism that allows for the mapping of different time bases. In Aïello et al.,[35] only a DEVS modeling scheme concerning both abstraction and time hierarchies is proposed. The simulation aspects have not been detailed and implemented. We can also point out that in the DEVS CD++ (http://sourceforge.net/projects/cdpptoolkit/) software environment, some time conversion functions (methods) offered by the C++ language are available.[36] These methods allow the handling of simulated time according to the format used in CD++ when dealing with different units (hours, minutes, seconds and microseconds). However, these methods are just used in order to convert the time into a time unit before any simulation. No time granularity management is involved in DEVS CD++ .

As already pointed out, in little previous work are the abstraction and time hierarchies associated with the SES formalism and treated as in the proposed approach: isolated as a concept on its own that can be formalized, integrated into the ontology of M&S using SES, and operationalized by leveraging the operations of pruning and transformation to DEVS. In Seck[37] the author proposed to extend the SES formalism in order to deal with multi-perspective concepts (which can be viewed as abstraction hierarchies). The approach depends on the formalization of the concept of ''aspect'' in order to add the possibility to define a kind of abstraction hierarchy. However no implementation nor any extension toward time granularity is mentioned. The work[30] also proposed an interesting approach based on the introduction of the ''aspect'' and ''bridges'' notions. The ''aspect'' notion can be understood as the possibility to extend DEVS (and as we propose SES) with the possibility to add a new hierarchy to DEVS (and SES for us). The ''bridges'' notion allows the specification of the transfer functions between levels. However no implementation is proposed.

We have to point out that there are, a priori, no links between time granularity and abstraction hierarchies. One can define models of a system that will involve time granularity and no abstraction hierarchy, and in a similar way a user may have to define a modeling scheme involving several levels of abstraction and only one time granularity. However, in both cases the objectives of abstraction hierarchies and time granularity are the same.

- Multi-level modeling (abstraction or granularity) for increased accuracy: In order to increase the accuracy of the modeling scheme, it is necessary to define models at lower levels of abstraction and time granularity.
- Approximate simulations for increased performance: Defining models of a system at different levels of abstraction or granularity will allow the performance of simulations with a lot of details (high levels of abstraction of granularity), which will require a lot of time-consuming or lower detail (low levels of abstraction or granularity), which will permit increases in the performance of the simulations in terms of CPU time (but of course with less accuracy).
- Approximate modeling scheme (at high levels of abstraction or granularity) for lower model development time: The design of models at high levels of abstraction or granularity allows the user to develop models faster than at lower levels.
- Interoperability between models at different levels of granularities or different levels of abstraction for reusability: If a user is using models at different levels of abstraction or granularity, he will be able to perform the simulations of interconnections of such models using the proposed approach.

3.2. Contributions

The paper concerns the extension of SES in order to deal with both abstraction and time hierarchies. The SES supports development, pruning, and generation of a family of simulation models. By introducing the possibility to define models at different levels of abstraction and different levels of time granularity (and associated transfer functions between levels), SES will support hierarchical composition in which families of models are generated and tested via pruning.

The approach presented in this paper depends on the definition of two atomic models (upward atomic model (UAM) and downward atomic model (DAM)), which are automatically inserted into a coupled model involving several levels of abstraction when the user has to perform the simulation at different levels of abstraction; no modification of the abstract DEVS simulator is required, and the functions allowing transition from one level of abstraction to another one are embedded into the DAM and UAM (these functions can be specified in the UAM and DAM code or in the SES corresponding to the coupled model to be simulated). The idea consisting of adding a new atomic model that allows the extension of the DEVS formalism is not new in the field of DEVS M&S.[34,38] Barros[34] introduces a new specific atomic model (named ''network executive'') that encodes in a state the information about the structure of a dynamic coupled model (named ''dynamic structure system network''). This concept has been proposed for the M&S of dynamic DEVS models where the structure of such models evolves over the simulation process. The DEVS coupling scheme of a coupled model depends on the selected state of the network executive atomic DEVS model embedded in the coupled model. Concerning the time granularity, the abovementioned approaches have brought a lot to this area of research (combining time granularity with DEVS). Aïello et al.[35] have done explorative work concerning time granularity, and proposed a formal approach for DEVS modeling of time hierarchies. However, they did not propose any solution for simulations of such DEVS models. Based on the specification of two new kinds of models (aspects and bridges), Seck and Honig[30] suggested that time granularities can be managed as a future extension of the proposed work.

We address the SES extension challenge by proposing two new typologies of the specialization notion. A SES represents the components of a system and their decompositions, taxonomies and couplings. The components are called ''entities'' and an entity may have several specializations, each representing a taxonomy of the possible variant of an entity. In addition, we have also defined how these two new taxonomies will have transfer functions constraints attached to them. Transfer function constraints allow the specification of the way information is transferred upward and downward between an entity and its variants.

We address the abstraction hierarchy challenge by proposing a generic approach in order to deal with abstraction hierarchies into DEVS. The proposed approach depends on the definition of two atomic models (DAM and UAM) that control the transfer of information downward and upward between levels of abstraction, which allows the simulation of a coupled model involving several levels of abstraction using an abstract DEVS simulator (Classical DEVS or PDEVS). We show in Section 3.4 how the DAM and UAM are automatically inserted in the coupled model that is going to be simulated, and how they allow the performance of the transfer functions that have been specified by the user in the framework of the SES (or directly using the DEVS formalism).

We address the time hierarchy challenge by proposing the same kind of approach in order to deal with the time granularity hierarchy. Once again, time information between levels of granularity will be performed using the two atomic models DAM and UAM. The user has to specify (by using SES transfer constraints or by writing the corresponding δ_{ext} function in DEVS) the way time information has to be considered when simulating a coupled model involving atomic models at different levels of granularity. We show in Section 3.4. how the DAM and UAM are used in this case of time granularity.

We have to point out that, in order to deal with both abstraction hierarchie and time hierarchie granularity, there is no real need for SES; it is possible to do the same without SES since the concept of abstraction hierarchie and time granularitie into coupled DEVS models. For example, in an implementation-centered approach (with the DEVSimPy framework, for example), a user will be able to define models at different levels of abstraction of time granularities and still perform the simulation of such models without using SES. The proposed approach to manipulate DEVS models at different levels of hierarchy depends on the definition and automatic generation of two types of DEVS atomic models that will allow the transfer of information (up and down) between levels. The abstract simulator is not modified, so that the presented approach fits with both the DEVS or its parallel variant PDEVS algorithm.

3.3. Modeling aspects

We introduce in this part the basic concepts for M&S of complex systems involving several levels of abstraction and several levels of time granularity in the framework of the SES and DEVS formalism. The three main features that have to be added to the DEVS formalism to reach this goal, are the following: (i) definition of different levels of abstraction – allowing the complexity of a system to be taken into account in a gradual way; (ii) definition of different levels of time granularity allowing modeling information of the considered system at different timebase to be taken into account; (iii) definition of a generic simulation approach allowing the processing of the simulation of systems, whatever the involved levels of abstraction or levels of time granularity.

When defining a model, a level of abstraction should be indicated. Dealing with M&S at different levels of abstraction in the DEVS formalism corresponds to the definition of coupled models at level of abstraction N that are composed by atomic models defined at level of abstraction N, and coupled models that can be defined at level of abstraction N + 1. When a coupled model at level N has a component that is a coupled model described at level N + 1, two kinds of functions must be added: (i) downward functions that allow the description of how the events (data) are transferred from level N to level N + 1 (disaggregation); (ii) upward functions that allow the description of how the events (data) are transferred from level N + 1 to level N (aggregation).

The simulation approach that we have defined is similar to the one proposed in the DEVS abstract simulator, as introduced by Zeigler.[3] The only difference consists in executing the downward/upward functions every time a data transfer between levels of abstraction has been defined in the modeling part by the user.

When defining a model, a level of time granularity should be indicated. Dealing with M&S at different levels of time granularity in the DEVS formalism corresponds to the definition of coupled models at level of granularity N that are composed by atomic models defined at level of time granularity N, and atomic models that can be defined at level of time granularity N + 1. When a coupled model at level N has a component that is an atomic model described at level N + 1, two kinds of functions must be added: (i) finer functions that allow the description of how events have to be expanded when going from level N to level N + 1; (ii) coarser functions that allow the description of how events have to be aggregated when going from level N + 1 to level N. As with the abstraction hierarchy, the simulation approach that we have defined is the similar to the one proposed in the DEVS abstract simulator with the difference consisting in executing the finer/coarser functions every time a time granularity conversion between levels of time granularity has been defined in the modeling part by the user.

Concerning the SES, two new types of specialization are proposed:

- The first one (notated *Abstraction* in Figure 7) has been defined in order to take into account abstraction hierarchy specifications (downward and upward functions) of the parent entity.
- The second one (notated *Time* in Figure 7) has been defined in order to take into account time granularity specifications (finer and coarser functions) of the parent entity.

Two unique features of the SES are decomposition and coupling. Decomposition describes how to decompose or breakdown an entity into entities. These entities can later represent components in a model constructed for the original entity. Coupling describes how information can flow among the components in the constructed model. Couplings appear in the outline of the SES; this display of couplings may be easier to scan through than the original natural language text. In order to specify the way transfer information has to be managed during the simulation, we have extended the natural language of SES using the following sentences.

- < variables > **are mapped downward/upward as** < other variables > **with** < a function > for specifying the transfer of information between variables belonging to different levels of abstraction.
- **Events on** < variables > **are more finely/coarsely expanded** for specifying the modification of temporal granularities.

In the next section we introduce the concepts to be used in order to realize the simulation of models that require abstraction hierarchies and time granularity.

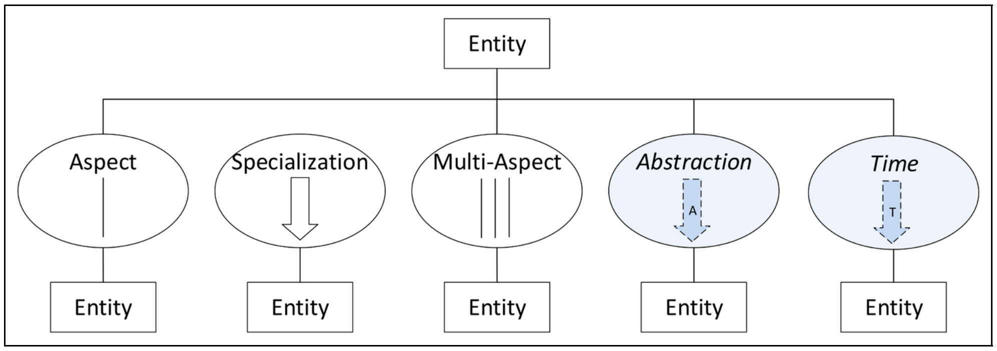

Figure 7. Integration of abstraction hierarchies and time granularity specifications into system entity structure (SES).

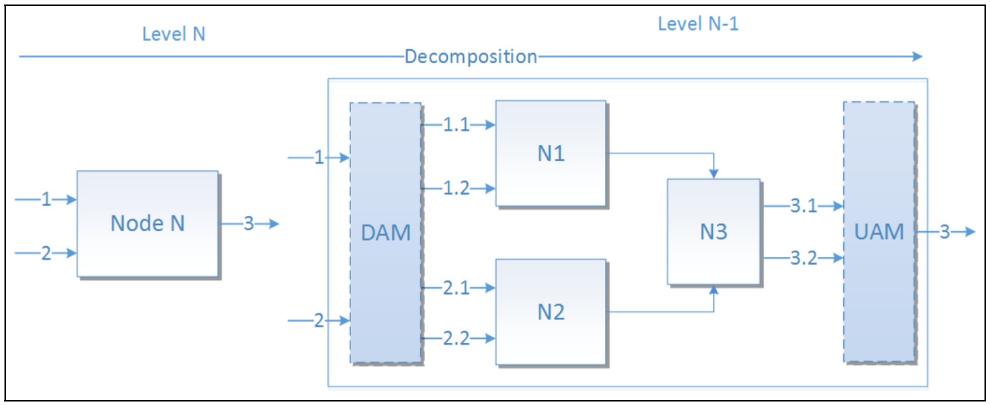

Figure 8. Additional downward atomic models (DAMs) and upward atomic models (UAMs) in the proposed approach.

3.4. Simulation aspects

The simulation approach that we have defined depends on the fact that the DEVS abstract simulator as introduced by Zeigler[3] has to be used without any modification. The goal is to be able to simulate DEVS models involving abstraction hierarchies using the classical or parallel simulator. We present below a way to perform the downward/upward functions every time a data transfer between levels of abstraction has been defined in the modeling part without any modification of the abstract simulator.

As with abstraction hierarchies, the simulation of DEVS models involving time hierarchies will be performed using the classical or parallel simulator. The goal is find a way to perform the finer/coarser functions every time a time granularity conversion between levels of time granularity has been defined in the modeling part without any modification of the abstract simulator.

In order to fulfill the above requirement, we propose to add two atomic models that will embed the transfer of information between levels of abstraction. These are: (i) the DAM, in order to transfer of information from a level of abstraction to a lower level, and (ii) the UAM in order to transfer information from a level of abstraction to a higher level (Figure 8). The atomic models used in this way extend the usual notion of DEVS transfer functions, which map between outputs and inputs in a memoryless manner. In contrast, the atomic models provide fully dynamic capabilities needed for time hierarchy mappings. Moreover, the introduction of the new facility enables: (i) explicit separation between the behavior of the models and the transfer functions of information (abstraction and granularity); (ii) the facilitation of the task of the user when writing the specification of the transfer functions.

This approach has the benefit of conserving the DEVS specifications, subject to the observance of the closure under coupling property. Another benefit to using atomic models in order to translate abstraction information between abstraction levels is that they simplify the time resolution between these levels. Moreover, the computational cost of the simulator is minimal since it will manage only two additional atomic DEVS models (DAM and UMA) rather than a new time-consuming tracking-based algorithm to support the compatibility between the interfaces of abstraction levels.

Furthermore, we propose to automatically add these two atomic models into a coupled model to be simulated. These two kinds of atomic models will be automatically introduced from the modeling part just before performing the simulation of models involving several levels of abstraction. Obviously, it would have to also be possible to do that statically. However, in order to facilitate the task of the user when he has to perform the simulation of DEVS models involving abstraction and/or time hierarchies, we propose to automatically add the UAM and DAM models into the coupled model that has to be simulated. The user will just have to define the upward and downward transfer functions before simulations, and the automatic addition of the DAM and UAM models will allow him to perform the simulations using the classical abstract simulator of DEVS or PDEVS formalism. This automatic generation could be also implemented during the SES pruning process, but it is not yet considered in this paper.

One approach to abstraction-level modeling is to add a behavior to the coupled DEVS model.[27] In contrast to this approach, and in order to fulfill the two requirements above, we propose to dynamically add two atomic models into coupled models to implement the transfer of information between levels of abstraction. These are (i) the DAM, in order to perform transfer of information from a level of abstraction to a lower level, and (ii) the UAM in order to perform the transfer of information from a level of abstraction to a higher level (Figure 8). These two kinds of atomic models will be automatically introduced from the modeling part just before performing the simulation of models involving several levels of abstraction. This approach has the benefit of conserving the DEVS specifications, subject to the observance of the closure and coupling property. Another benefit of using atomic models in order to translate abstraction information between abstraction levels is that they simplify the time resolution between these levels. The introduction of the two kinds of atomic model, DAM and UAM, allows the use of the classical or parallel abstract simulator concepts when performing simulation of models through several levels of abstraction.

Concerning a DAM atomic model, the transfer function is executed each time an input event is received on the input of the DAM model using the external transition, and output events on each one of the output ports of the DAM are immediately generated according to the transfer function (time advance = 0). Concerning the UAM model, the transfer function is executed each time each input of the UAM model has received an event. An output event is generated on the output port of the UAM model at a time defined using the UAM time advance function. One important feature to point out is that the DAM (respectively UAM) should be the first (respectively the last) in the priority list of the coupled model including DAM and UAM. This is done automatically before the simulation process. Obviously, this constraint will disappear using PDEVS[3] instead of classical DEVS. In PDEVS, the omission of the select function means that colliding models should transition in parallel instead of sequentially. This increases the possibility for parallelization, since the output function and transition functions can now happen in parallel by merging different bags.[39] Furthermore, as we can see in Figure 8, all the events pass through DAM/UAM and there are no conflicts with the external transition functions of all other models.

Let us now consider time hierarchies. Modeling according to several levels of time granularity is enabled by adding into a given coupled model, whose behavior is described according to a given timebase, an atomic model whose behavior requires execution at a finer or coarser timebase. In order to perform the simulation of models described according to different levels of time granularity, we have to define a mechanism that allows the conversion of the current timebase into a finer (respectively coarser) one when going downward – from a level of granularity N to a level N + 1 – (respectively upward – from a level of granularity N + 1 to a level N). The two atomic models presented above (DAM and UAM) are used in order to perform the time granularity conversions. The timebase conversion function allowing conversion from a given time granularity level to a finer one is associated with the DAM atomic model. The timebase conversion function allowing conversion from a given time granularity level to a coarser one is associated with the UAM atomic model. The modeler has to write the time granularity finer conversion function that expresses how an event message received at time t at the granularity level N is expanded into several event messages at level N + 1 (for example, as soon as an event arrives at the input of the model at level N, say at time t on the timebase Day (Figure 8), the time granularity finer conversion function associated with the DAM model is activated and allows the generation of 24 events planned at time $t + \frac{1}{24}$, $t + \frac{2}{24}$, …, $t + 1$ on the timebase Hour (Figure 8)). In the same way the modeler also has to write the time granularity coarser conversion function that expresses how a set of events collected at level N + 1 is aggregated into an event at level N. For example, as soon as 24 events are collected at the inputs of the UAM model on the timebase Hour, the time granularity coarser conversion function is activated and allows the aggregation of the 24 events into one event at time t on the timebase Day at level N.

4. Implementation

This section shows how it is possible to model the behavior of a watershed with several abstraction levels and several time granularity levels in the DEVSimPy framework. This subsection deals with a case study that is intended to illustrate and validate the concepts of abstraction and time hierarchies presented in Section 3. The case study concerns the behavior of a watershed, and will involve a SES modeling whose pruning will allow a set of DEVS models to be obtained according to different levels of abstraction and temporality. The case study is described in Section 4.2, the SES modeling (respectively DEVS modeling) in Section 4.3 (respectively 4.4). The simulation results are given and analyzed in Section 4.5.

4.1. Abstraction and time levels in DEVSimPy

When the user wants to implement an abstraction level in a coupled model with DEVSimPy: (i) he/she defines the level of abstraction using the spin control (see the black box in Figure 9); (ii) he/she specifies a behavior of two atomic models that control the downward/upward abstraction rules for a coupled model. The code of these two atomic models can be edited by clicking in the down (respectively up) arrow button for the DAM (respectively UAM) model. These atomic models will be instantiated and connected dynamically into the coupled model at the beginning of the simulation. The user does not need to know other functions (besides the transition function of the DEVS) for implementing abstraction-based models. Figure 9 depicts the CM0 coupled model at abstraction levels 0 and 1. The details of level 1 are shown in the detached frame, which is composed by: two atomic models (AM0 and AM1), three input ports (IPort 0, 1 and 2) and two output ports (OPort 0, 1).

When the user wants to implement a time granularity level in a coupled model with DEVSimPy: (i) he/she defines the level of time granularity using the spin control

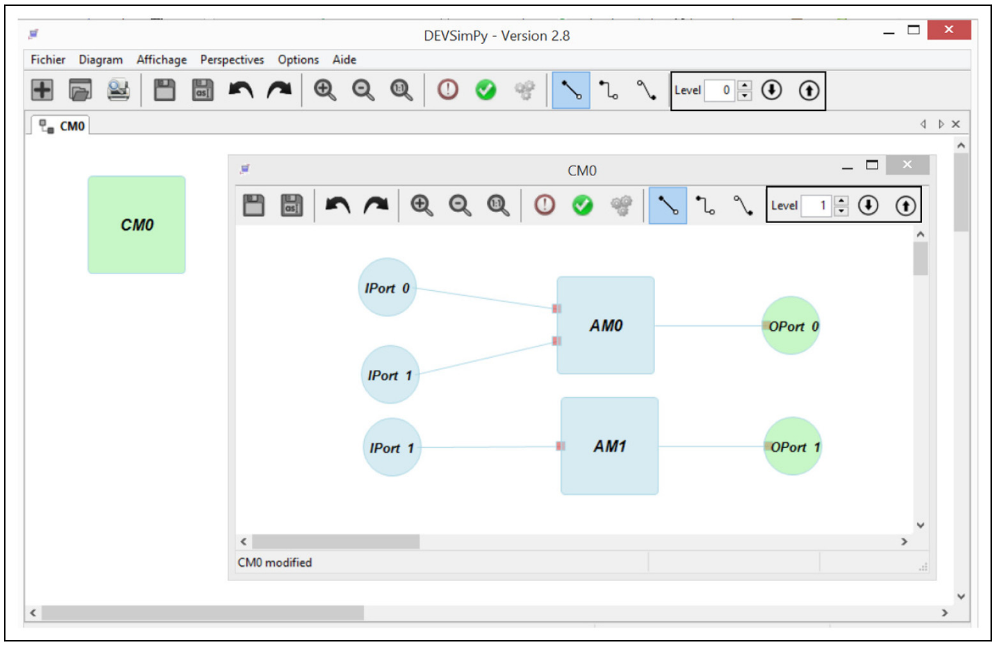

Figure 9. Abstraction levels of the CM0 coupled model into DEVSimPy.

(see black box in Figure 9); (ii) he/she specifies the behavior of the two atomic models (DAM and UAM) that control the finer downward/coarser upward time granularity rules for a coupled model. As for the abstraction level, the code of these two atomic models can be edited by clicking in the down (respectively up) arrow button for the DAM (respectively UAM) model. As in the case of the abstraction hierarchy management, these atomic models are instantiated and connected dynamically into the coupled model at the beginning of the simulation.

Listing 1 gives an example of the code of a DAM model. This code is partially generated using a template, and the user has just to code the transition functions and the time advance function. It allows the finer downward rules definition for both the abstraction hierarchy and the time granularity for a coupled model. The DAM atomic model has one input port (such as a port of an atomic model at level N) and when an event occurs, the external method is triggered and the message is stored in the ’msg’ variable (line 14). The DAM model sends two messages (since the atomic model at level N + 1 requires two ports) by multiplying the received message by 10 and 20 (lines 21, 22) at fixed time intervals $0.0416 = \frac{1}{24}$ (line 34) in any 24 periods (line 27). It should be pointed out that the time step of the DAM model at level N + 1 must be a multiple (or strictly, a least multiple) of the time step at level N. An UAM model can be specified in the same way. Regarding to the DAM of the Listing 1, an UAM model could send message(s) by combining all of the messages received during the period. The rule that defines the combination of the messages could be specified in the *UAM outputFunction* before sending the message(s). The UAM model goes to the ’SENDING’ state only when it receives all messages.

4.2. *Case study: Watershed behavior informal description*

An informal description of the hydrologic behavior of a watershed involving two kinds of abstraction specialization and one example of temporal specialization is given in this section. This case study illustrates the use of both abstraction and time hierarchies in the context of a real case application. The considered example concerns a watershed belonging to a mountainous part of France (the Alps).[40] It involves both liquid (rain) and solid (snow) precipitation over the period of a year.

Precipitation can be classified according to its nature, solid or liquid. Depending on the time of year and the geographical area concerned, the amount of solid precipitation will be different, moving between entirely solid (only snow) and entirely liquid (only in the form of rain). When precipitation falls as snow, the hydrological response of the watershed is not the one observed in case of rain: there is a lack of response in the short term due to the accumulation of solid precipitation in the form of snow cover at the soil surface. Then, when the conditions of melting are met (which can occur several days to several months after the occurrence of precipitation), this water is remobilized. It causes a delayed reaction of the watershed.

The behavior of the watershed is obtained by studying the rainfall data (expressed daily) and temperature data (expressed daily and hourly). Different models defined at

Listing 1. Example of Python class for a DAM DEVS atomic model.

```
class DAM(DomainBehavior):
### DEVS Class for DAM model

def _ _init_ _(self):
   ### Constructor
   DomainBehavior._ _init_ _(self)
   self.state = {'status': 'IDLE', 'sigma':INFINITY}
   self.msg = None
   self.value = None
   self.cpt = 0

def extTransition(self):
   ### DEVS external transition function
   self.msg = self.peek(self.IPorts[0])
   self.state['sigma'] = 0 if self.state['status'] == 'IDLE' else INFINITY
   self.state['status'] = 'SENDING' if self.state['status'] == 'IDLE' else 'IDLE'

def outputFnc(self):
   ### DEVS output function
   self.value = self.msg.value[0]
   self.poke(self.OPorts[0], Message([self.value*10,0,0], self.timeNext))
   self.poke(self.OPorts[1], Message([self.value*20,0,0], self.timeNext))

def intTransition(self):
   ### DEVS internal transition function
   self.cpt +=1
   if self.cpt == 23:
         self.state['status'] = 'IDLE'
         self.state['sigma'] = INFINITY
         self.msg = None
         self.cpt = 0
else:
         self.state['status'] = 'SENDING'
         self.state['sigma'] = 0.0416

def timeAdvance(self):
   ### DEVS Time Advance function
   return self.state['sigma']
```

different levels of abstraction or time granularity are defined and simulated. The simulation results of these models are then compared with the observed steam-flow of the watershed.

The hydrologic behavior can be specified in four different ways.

- At the highest level of abstraction (level 0), the behavior is the simplest one, just considering a percentage of the rainfall.
- At a lower level of abstraction (level 1), the behavior is expressed by taking into account the altitude parameter (using a *Montain* atomic model) and three hydrogeological layers (soil layer, surface layer and aquifer layer).
- At another lower level of abstraction (level 2) the *Montain* model is detailed in order to take into account the snow effect (at the highest level of temporality (level 0) the snow effect is considered on a daily basis).
- At the same level of abstraction (level 2), the *Montain* model is then detailed at a lower level of temporality (level 1) on an hourly time basis.

The next section presents the SES modeling of the previously introduced hydrological behavior of a watershed.

4.3. SES modeling

Figure 10 depicts the SES of the WS (WaterShed) model with the use of both abstraction hierarchy and time hierarchy specifications. WS0 and WS1 are abstract specializations of the WS entity, and level mapping specifications are introduced. The dashed lines with elements like $(<\text{model } 1>, <\text{model } 2>, <\text{port } 1>, <\text{port } 2>)$ are added in order to hold the information of the coupling

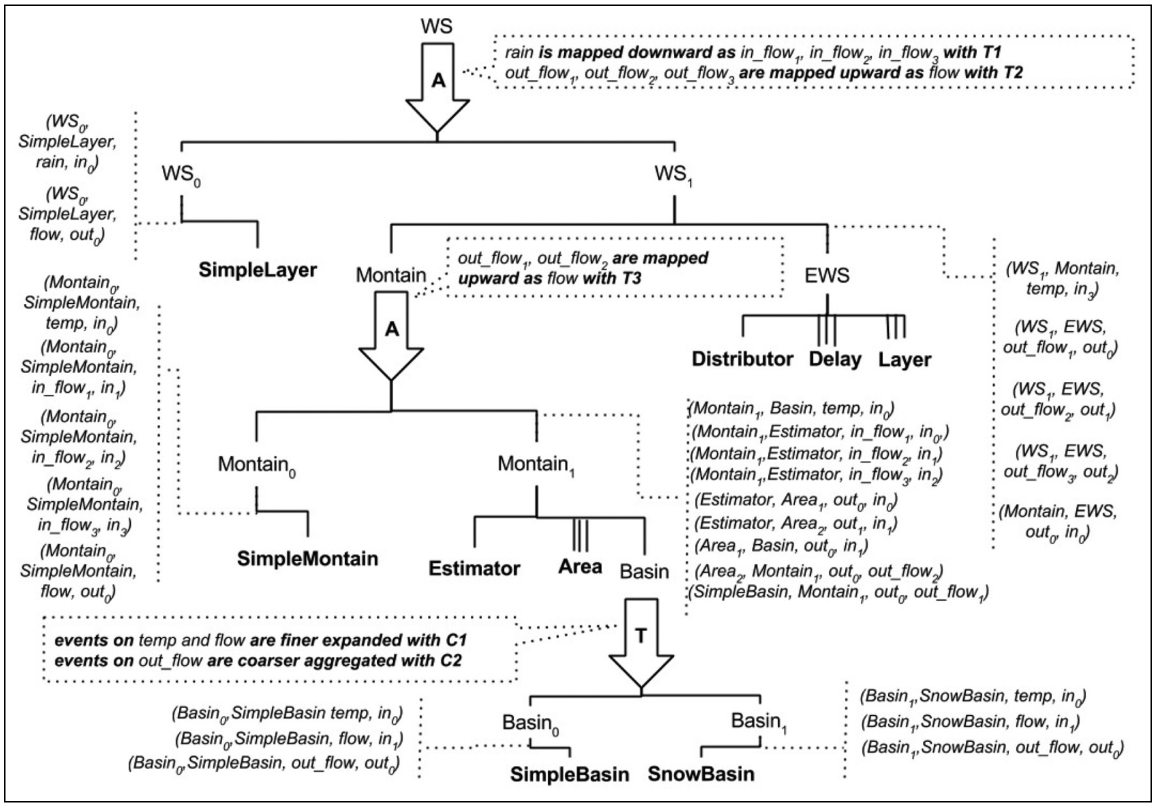
WS
A
rain is mapped downward as in_flow1, in_flow2, in_flow3 with T1
out_flow1, out_flow2, out_flow3 are mapped upward as flow with T2
WS0
WS1
(WS0, SimpleLayer, rain, in0)
(WS0, SimpleLayer, flow, out0)
SimpleLayer
Montain
out_flow1, out_flow2 are mapped upward as flow with T3
EWS
Distributor
Delay
Layer
(WS1, Montain, temp, in3)
(WS1, EWS, out_flow1, out0)
(WS1, EWS, out_flow2, out1)
(WS1, EWS, out_flow3, out2)
(Montain, EWS, out0, in0)
(Montain0, SimpleMontain, temp, in0)
(Montain0, SimpleMontain, in_flow1, in1)
(Montain0, SimpleMontain, in_flow2, in2)
(Montain0, SimpleMontain, in_flow3, in3)
(Montain0, SimpleMontain, flow, out0)
Montain0
Montain1
SimpleMontain
Estimator
Area
Basin
(Montain1, Basin, temp, in0)
(Montain1, Estimator, in_flow1, in0)
(Montain1, Estimator, in_flow2, in1)
(Montain1, Estimator, in_flow3, in2)
(Estimator, Area1, out0, in0)
(Estimator, Area2, out1, in1)
(Area1, Basin, out0, in1)
(Area2, Montain1, out0, out_flow2)
(SimpleBasin, Montain1, out0, out_flow1)
events on temp and flow are finer expanded with C1
events on out_flow are coarser aggregated with C2
T
Basin0
Basin1
SimpleBasin
SnowBasin
(Basin0, SimpleBasin temp, in0)
(Basin0, SimpleBasin, flow, in1)
(Basin0, SimpleBasin, out_flow, out0)
(Basin1, SnowBasin, temp, in0)
(Basin1, SnowBasin, flow, in1)
(Basin1, SnowBasin, out_flow, out0)

Figure 10. System entity structure (SES) of the Watershed behavior.

between entities. In this way, the level mapping abstract specifications could be invoked with the downward and upward functions as:

1. *rain* is mapped downward as *in_flow*$_1$, *in_flow*$_2$, *in_flow*$_3$ with "T1", where *rain* is the input port of "WS", *in_flow*$_1$, *in_flow*$_2$, *in_flow*$_3$ are the input ports of *WS*$_1$ and "T1" is the downward function.
2. *out_flow*$_1$, *out_flow*$_2$, *out_flow*$_3$ are mapped upward as flow with "T2", where *flow* is an output port of "WS", *out_flow*$_1$, *out_flow*$_2$, *out_flow*$_3$ are output ports of *WS*$_1$ and "T2" is the upward function.

In the same way, *Montain*$_0$ and *Montain*$_1$ are abstract specializations of the *Montain* entity. The upward function is described as follows: *out_flow*1, *out_flow*2, *out_flow*3 are mapped upward as flow with "T3", where flow is the output port of *Montain*, and *out_flow*$_1$, *out_flow*$_2$ and *out_flow*$_3$ are the output ports of *Montain*$_1$. We have to highlight the fact that the transfer of information between: (i) the input ports *temp* (respectively *in_flow*$_1$, *in_flow*$_2$ and *in_flow*$_3$) of *Montain* and the input port *in*$_0$ (respectively *in*$_1$, *in*$_2$, *in*$_3$) of the *Montain*$_1$ entity are mapped with the identity function (they are not indicated in Figure 10).

Figure 10 also introduces a temporal specialization associated with the entity called *Basin*. *Basin*$_0$ and *Basin*$_1$, which are temporal specializations of the *Basin* entity. The downward and upward temporal transfer functions are described as follows: (i) events on *temp* and *flow* of *Basin* are more finely expanded with "C1" where *temp* and *flow* are the inputs ports of the *Basin* model, and "C1" is the conversion function of information between *Basin*$_0$ (level 0 of temporality) and *Basin*$_1$ (level 1 of temporality); (ii) events of "out_flow" are coarsely aggregated with "C2" where "out_flow" is the output port of the entity *Basin* and "C2" is the conversion function between entity *Basin*$_1$ (level 1 of temporality) and entity *Basin*$_0$ (level 0 of temporality).

Before detailing the DEVSimPy implementation, we provide a short overview of the approach to developing the family of models for the WS. Using the pruning operation involved in the SES modeling scheme, a set of intersecting SESs representing a suite of related families of models is derived. In such a suite, there are SESs that are components of other SESs. A SES is a component of another SES in the sense that the models that the first SES generates are components of models generated by the second SES.[8] Four families can be derived.

- "*WS*$_0$" family based on the *SimpleLayer* model.
- "*WS*$_1$ with *Montain*$_0$" family that involves the EWS SES and the *SimpleMontain* model.

- "WS_1 with $Montain_1$ and $Basin_0$" family that depends on the *EWS* SES, the $Montain_1$ SES involving the *SimpleBasin* model.
- "WS_1 with $Montain_1$ and $Basin_1$" family that depends on the *EWS* SES, the $Montain_1$ SES involving the *SnowBasin* model.

The user has to define the five transfer functions involved in Figure 10.

1. T1: which allows the downward transfer of information between the input port rain at the level 0 abstraction of the WS component and the input ports in_flow_1, in_flow_2 and in_flow_3 at the level 1 abstraction of the WS component. This function is defined as follows: in_flow_1 (respectively in_flow_2 and in_flow_3) = 60% (respectively 30% and 10%) of rain.
2. T2: which allows the upward transfer of information between the output ports out_flow_1, out_flow_2 and out_flow_3 at the level 1 abstraction of the WS component and the output port *flow* at level 0 abstraction of the WS component. This function is defined as follows: $flow = out\_flow_1 + out\_flow_2 + out\_flow_3$.
3. T3: which allows the upward transfer of information between the output ports out_flow_1 and out_flow_2 at level 1 abstraction of the *Montain* component and the output port *flow* at level 0 abstraction of the *Montain* component. This function is defined as follows: $flow = out\_flow_1 + out\_flow_2$.
4. C1: which allows the downward transfer of information between the input ports *temp* and *flow* at level 0 temporality of the component *Basin* and the input ports *in*0 and *in*1 at level 1 temporality of the component *Basin*. This function is defined as follows: an event at time t (daily basis) is expanded into 24 events at time $t + \frac{1}{24}$, $t + \frac{2}{24}$, $t + \frac{3}{24}$, …, $t + 1$ (hourly basis).
5. C2: which allows the upward transfer of information between the output port *out*0 at level 0 temporality of the component *Basin* and the output port *out_flow* at level 0 temporality of the component *Basin*. This function is defined as follows: it performs the sum of the values of each event arriving on port "out0" between time t and time t + 1 (hourly basis) in order to define an event at time t + 1 (daily basis) whose value is the previously computed sum.

The other transfer functions are not highlighted in the paper since they correspond to the identity function. The next section presents the DEVS models associated with the previously introduced SES model and modeled into DEVSimPy framework.

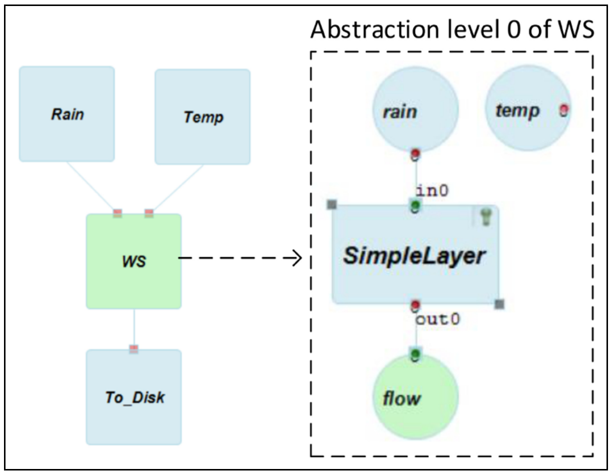

Figure 11. Watershed at abstraction level 0 in DEVSimPy.

4.4. DEVSimPy modeling

A description of the DEVS modeling scheme of the watershed behavior involving three levels of abstraction and two levels of temporality is given in this section. We call WS a coupled model describing the behavior of a watershed at a level 0 abstraction. It has two inputs corresponding respectively to the daily temperature (*temp*) and the rain precipitation (*rain*), and an output (*flow*) corresponding to a quantity of water coming from the watershed.

Figure 11 details the WS coupled model describing the behavior of a watershed at level 0 and is composed by one atomic model (*SimpleLayer*) at level 0. The behavior of the *SimpleLayer* is as follows: 20% of the water that arrives as input is generated on the output. The description model of watershed behavior at the abstraction level 1 is given in Figure 12. It is a coupled model integrating two other coupled models: the *Montain* model and the *EWS* (Elementary WaterShed) model. The behavior of the *EWS* model is described using a classical DEVS coupled model as shown in Figure 12 while to be able to define the behavior of the *Montain* model two different levels of abstraction have been defined.

A *EWS* model (Figure 12) receives one input *Alt* corresponding to the flow coming from the *Montain* model. An *EWS* model allows computation on its outputs – the flows associated with the three layers (*flow*1, *flow*2, *flow*3) that characterize it. The behavior of the atomic models involved in the *EWS* coupled model are not described in this paper since it is a simple DEVS coupled model (without any abstraction or time hierarchies).[41] As mentioned above in the *WS* coupled model at level 1, the *EWS* model

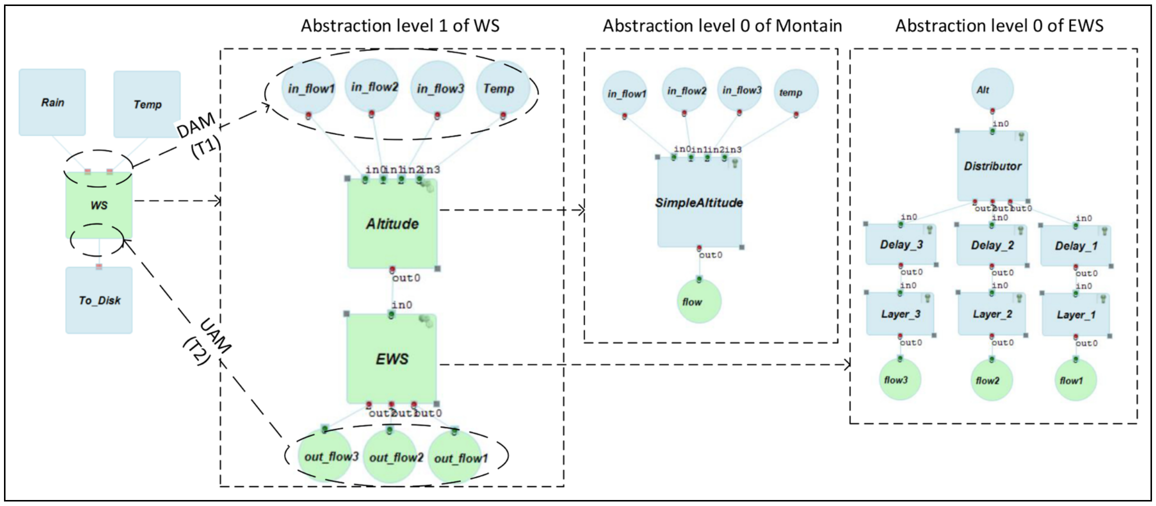

Figure 12. Watershed at abstraction level 1 with $Montain_0$ in DEVSimPy.

is interconnected with the *Montain* model (Figure 12): the precipitation input of the *EWS* model (called *Alt*) is connected to the output of the *Montain* model out_0.

In order to specify the behavior of the *Montain* model we have again used the SES abstraction specialization as described in Figure 10. The *Montain* entity of the SES is described at level 0 abstraction by the coupled model $Montain_0$ and at level 1 abstraction by the coupled model $Montain_1$. Figure 12 details the coupled model describing the behavior of the *Montain* model at level 0. It has one input *temp* corresponding to the daily temperature, and three other inputs in_flow_1, in_flow_2 and in_flow_3 corresponding to the daily precipitations at different points of the watershed, while it has one output (*flow*) corresponding to a quantity of water coming from *Montain* model. The $Montain_0$ coupled model is composed by one atomic model (*SimpleMontain*) at level 0 (Figure 12). The input port in_flow_1 (respectively in_flow_2, in_flow_3, *temp*) of $Montain_0$ is connected to the input in_0 (respectively in_1, in_2 and in_3) of the *SimpleMontain* atomic model. The behavior of the *SimpleMontain* is as follows: the output of the *SimpleMontain* model (out_0) is computed by performing 10% of the sum of the three inputs in_flow_1, in_flow_2 and in_flow_3.

The $Montain_1$ is a coupled model for estimating rainfall depending on the altitude (and the effect of snow). This model requires the definition of two kinds of atomic models (Figure 13):

- *Estimator* that calculates the estimated rainfall at each point of the mesh.
- *Area* that calculates the rainfall on the considered area linked with the *Area* atomic model. In the considered example we have to deal with two different areas (*Area_1*, *Area_2*) and only the *Area_1* has to deal with the snow effect.

The behavior of the *Estimator* is that, every time events (vi, t) are received in the three inputs (in_flow_1, in_flow_2, in_flow_3, it performs the sum S of the three values of the events vi and sends an event ($S * \frac{2}{3}$, t) from output *out*0 (connected to $Area_1$) and an event ($\frac{S}{3}$, t) from output *out*1 (connected to $Area_2$).

The behavior of the *Area* model is as follows: the *Area* atomic model allows the modeling of the behavior of parts of the watershed according to the altitude. In the considered example, $Area_1$ corresponds to the high part of the watershed and is connected with the *Montain* model while $Area_2$ corresponds to the lower part of the watershed and is not connected with a *Montain* model. When the altitude is high ($Area_1$ in our case) only a small (respectively great) part of the received flows is going to be found in the river flow, reflecting the storage of the water in the form of snow (respectively the snow melting effect). The behavior is as follows: when an input event (v,t) is received at the *in*0 input of an *Area* atomic model, an output event ($v * \frac{3}{4}$, t) is sent to the output *out*0 of the *Area* atomic model. As explained above, the *out*0 output port of the $Area_1$ model is connected to the *in*0 input port of the *Basin* model in order to take into account the snow effect.

Snow is another important element that can substantially alter the water balance from one month to another, especially in mountainous areas. For that it is necessary to:

- compute the proportion of rain transformed into snow based on the data of a reference thermometer station;
- manage the effect of the snow, which means storing the snow when the temperature is lower than 0°C and compute the quantity of water that will be generated from the stored snow when the temperature is greater than 0°C.

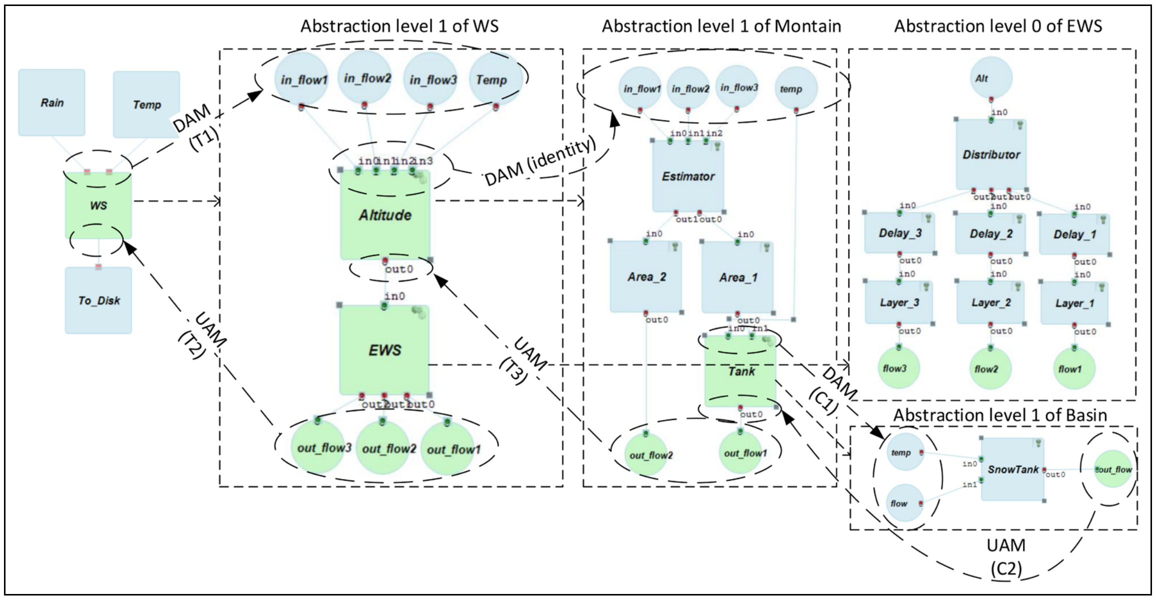

Figure 13. Watershed at abstraction level 1 with *Montain*$_1$ and *Basin*$_1$ in DEVSimPy.

Thus, in order to take into account the snow, from rainfall data (daily data) and temperatures (hourly data) we add a new type of component named *Basin*, which is connected with the area component corresponding to the highest part of the watershed.

In the example of Figure 10 we notice that the *Basin* entity is temporally specialized into two different entities: *Basin*$_0$ and *Basin*$_1$. Therefore, the *Basin* component can be described using two possible behaviors, located on two different time granularity levels (temporal level 0/1 is Day/Hour). Data exchange between these two temporal levels is provided by two transfer conversion functions C1 and C2. Considering the granularity level 0, the input data are processed using a Day timebase unit, which is coarser than the timebase, used at the level 1 (Hour timebase unit). The behavior of the *Basin* entity is therefore less precise at temporal level 0 than at temporal level 1 when the number of data taken into account is greater (information per hour instead of information per day). This implies a greater accuracy of results at temporal level 0, but a longer period of simulation in relation to the temporal level 1. When performing the pruning using the SES formalism, the user will have to choose between the two possible behaviors, obviously knowing that obtaining highly accurate results is done at the expense of simulation time, because it increases (sometimes considerably).

At temporal level 0, the *Basin*$_0$ model has an input called *in*0 representing the observed daily temperatures and an input called *in*1 representing the observed rainfall. It also has an output called *out*0 representing the quantity of water (*flow*) coming out from the *Basin* model by taking into account the effect of the snow.

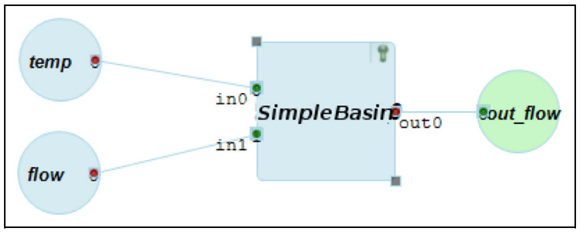

Figure 14. *Basin* coupled model at time level 0.

Figure 14 details the coupled model describing the behavior of the *Basin* model at level 0. It is a coupled model involving one atomic model called *SimpleBasin* that has two state variables called *Stored* (which is used to model the storage of the snow) and *Melting* (which is used to model the melting). The external transition can be informally specified as follows: if the input temperature is negative, then the quantity of water of the rain input is added to the *Stored* state variable, while if the input temperature is positive, the *Stored* state variable is decreased from a certain value representing the melting of the snow (this value is a percentage of melting obtained according to the value of the temperature) and the *Melting* state variable is set to the previous computed quantity. The output of the *SimpleBasin* model (flow) is set at the value given by the *Melting* state variable (using the lambda output function) while the internal transition allows the Melting state variable to be reset to zero.

Figure 13 depicts the *Basin*$_1$ coupled model. It involves the atomic model called *SnowBasin*. The description is similar to the previous description. The only difference is that, when events are arriving on the inputs *in*0 and *in*1 of the *SnowBasin* component, the level of temporality is different (instead of a daily time step, it is now an hourly time step that has to be considered). Once again, DAM and UAM atomic models are inserted in order to perform the conversion between the two time granularities. The DAM is used in order to split the events arriving at the inputs at time t into 24 events at different time units respectively $t + 0.025$, $t + 0.05$, $t + 0.075$, $t + 0.1$, $t + 0.125$, …, $t + 1$. Similarly the UAM atomic model is used to aggregate the 24 last events arriving from the output port out0 into one event at time t + 1.

The next section details how the transfer of information between levels of abstraction and levels of temporality are performed in the case of the previously defined DEVS models.

4.5. DEVSimPy simulation results

In this section we present the simulation results obtained after the pruning of the SES of Figure 10. This pruning results in four different configurations.

1. WS_0, which corresponds to the level of abstraction 0 of the *WS* component involving the atomic model *SimpleLayer*
2. WS_1 with $Montain_0$, which corresponds to the level of abstraction 1 of *WS* where the component *Montain* is described at level of abstraction 0
3. WS_1 with $Montain_1$ and $Basin_0$, which corresponds to the level of abstraction 1 of *WS* where the component *Montain* is described at level of abstraction 1 with the component *Basin* described at level of temporality 0
4. WS_1 with $Montain_1$ and $Basin_1$, which corresponds to the level of abstraction 1 of *WS* where the component *Montain* is described at level of abstraction 1 with the component *Basin* described at level of temporality 1

The left-hand side of Figure 11 gives the context of the experiment: two generator atomic models (*Rain* and *Temp*) allow the sending of the amount of rain and temperature of each day of a year concerning the watershed under study (situated in the French Alps). The *WS* coupled model corresponds to the models described previously in the SES of Figure 10. According to the pruning procedure of the SES, the *WS* model can be replaced by the four configurations listed above. The DEVS simulations of the *WS* model have been performed for the four previously DEVS models. The results of the simulation are expressed from Figure 15 thanks to the *To_Disk* atomic model. Each of these figures presents a time series comparing simulated and observed daily flows for the one-year period (365 days for the year 2009) when using each of the four previously mentioned DEVS models.

The top-left-hand side of Figure 15 depicts the simulation result with the *WS* model at the abstraction level 0. The shapes of the two curves are broadly similar, with some additional peaks due to the lack of details in the behavior of the model. The simulation results depicted on the top-right-hand side of Figure 15 are not significant, despite the increasing of the abstraction level for *WS*. This is due to the fact that more details (abstraction and granularity) are required. Therefore, the bottom-left-hand side of Figure 15 shows an improvement of the results when another level of abstraction is introduced with the *Montain* model. The amplitude of the simulated results is close to the observed streamflow due to the introduction of the models *Area* and *Basin*, which allows the consideration of the effect of the temperature on a daily basis. The bottom-right-hand side of Figure 15 points out the effect of the temporal granularity on obtained results. Simulated peaks are closer (days 197-201) to the observed streamflow peaks when the snow effect on an hourly time basis is considered.

Furthermore, in order to perform a better comparison between the four different simulations, we propose to use the Pearson correlation coefficient (PCC).[42] Correlation is a technique for investigating the relationship between two quantitative, continuous variables, for example, age and blood pressure. Pearson's correlation coefficient is a measure of the strength of the association between the two variables. It measures the dependence between two variables X and Y, giving a value between $+1$ and -1 inclusive, where 1 is total positive correlation, 0 is no correlation, and -1 is total negative correlation. It is widely used in the sciences as a measure of the degree of linear dependence between two variables. We have computed the PCC between the 365 values of the observed streamflow and the 365 values of each of the four obtained results. The PCC between observed and simulated streamflow quantifies the agreement between the two records. If the estimated series were a perfect prediction of the observations, the PCC would be one. The best predictions correspond to the ones that are the closest to one.

The distribution of the normalized PCC and CPU time for each of the four presented DEVS models is shown in Table 2. It is immediately apparent that each time a lower abstraction level or a lower time granularity level is considered, the PCC is improved and the CPU time increases. From this measure, the last model involving the hourly basis simulation for the *Montain* model appears more competitive than the three DEVS models working at the same time granularity level. In the same way, each time a lower level of abstraction has been used, the PCC is better. From these results we have pointed out the efficiency of a modeling scheme involving both abstraction hierarchies and time granularity. Obviously, the PCC can be improved by refining the DEVS model, but that was not the goal of the proposed work.

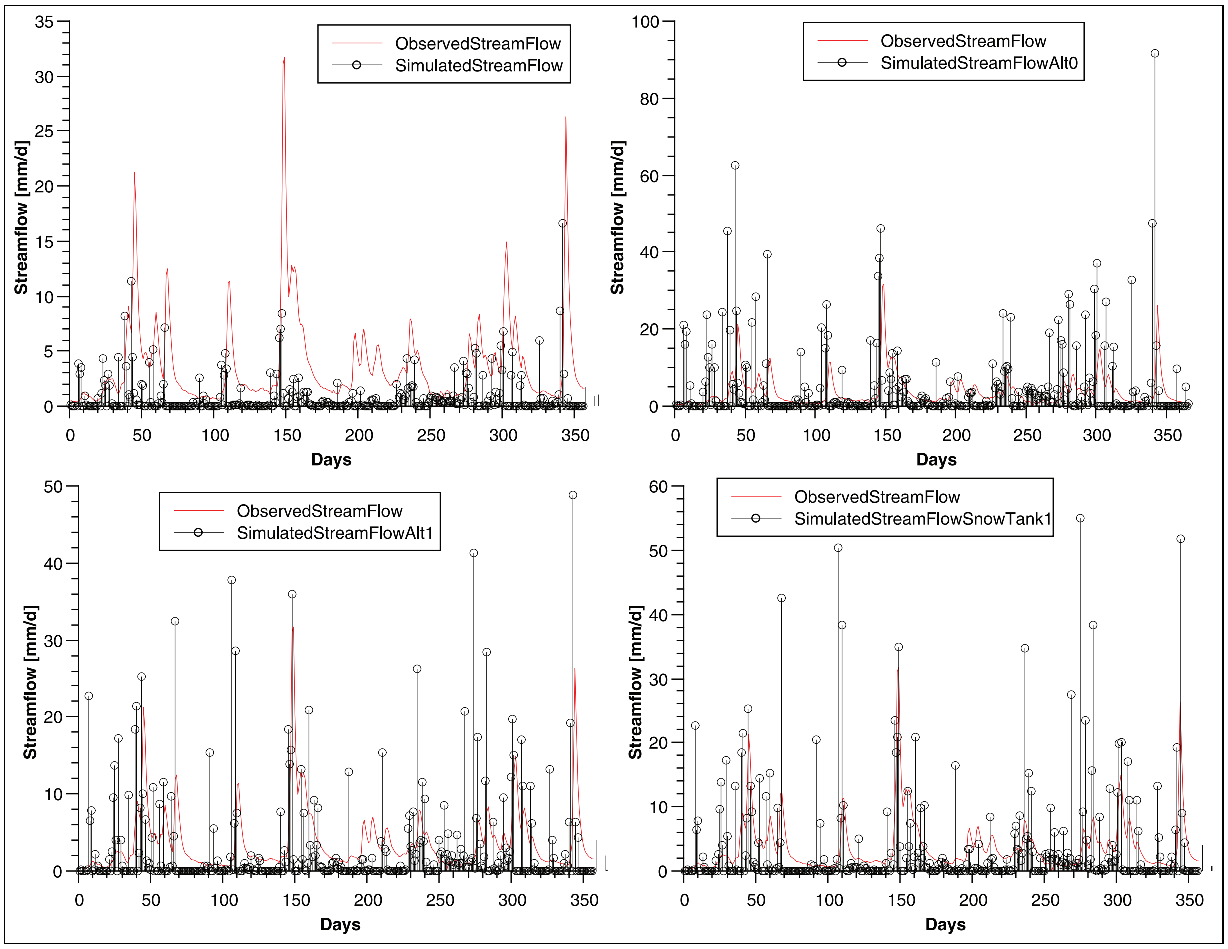

Figure 15. Simulation results at different level of abstraction and time granularity (top-left-hand side for WS_0, top-right-hand side for WS_1, bottom-left-hand side for WS_2, bottom-right-hand side for WS_3).

Table 2. Pearson correlation coefficient (PCC) and CPU time comparison.

	PCC	CPU (s)
Simulated StreamFlow	0.072649636	8.345
Simulated StreamFlow (Alt0)	0.073450706	8.367
Simulated StreamFlow (Alt1)	0.302929338	9.568
Simulated StreamFlow (SnowTank1)	0.475553476	11.567

5. Conclusion and future work

This paper deals with the M&S of systems described according to several levels of abstraction and levels of granularity. The notion of hierarchical levels of abstraction and levels of time granularity have been introduced and compared with the notion of hierarchical levels of description. An extension of SES has been proposed in order to integrate both abstraction and time granularity hierarchies in M&S. We have informally detailed how to deal with the notion of levels of abstraction and time granularity in the framework of the DEVS formalism. The presented approach allows the simulation of models involving several levels of abstraction and several levels of time granularity through the association of downward and upward functions to two new atomic models: DAM and UAM. The partial implementation performed using the DEVSimPy framework has been validated on a real case of catchment basin behavior described using the extension of SES. This research opens up many directions of research and development. (i) Define and implement the capabilities entailed by abstraction hierarchies and time granularity theory including support for pruning the extended SES and transforming pruned entity structures into DEVS models. Theoretical support for developing abstractions and analyzing their error properties can also be integrated into the proposed. For example, the division of space into snow and rain regions can be justified with the uniformity condition requirements of the theory in Zeigler,[2] Chapters 13,14. (ii) It is possible that the modeler needs to consider abstraction and time hierarchies all at once. SES has to be extended in order to merge the two kinds of hierarchy. (iii) Emergence[29]: we plan to use both the abstraction and time hierarchies in order to deal with emergence of DEVS complex systems. The proposed approach fits very well in the framework Stimergic-DEVS and DEVS-CAS.[29] (iv) SES extension in MS4Me (www.Ms4systems.com)[1,43]: the goal is to show the generic nature of the approach by generating DEVS models from the SES in the MS4Me software based on the concepts proposed here. (v) Investigation around of the concept of superdense time trajectories[44]: we plan to investigate the notion of superdense time trajectories in the case of the management of time granularity in DEVS simulations. (vi) Incorporating recent advances in DEVS and SES developments: we envision adding the state-based (in contrast to behavior-based) hierarchy starting with the probabilistic hierarchies based on FP-DEVS (finite probabilistic DEVS) and Markov modeling (The MS4 modeling environment provides a suite of tools that support a probabilistic extension of FD-DEVS[45]); also the SES functions introduced recently by Pawletta et al.[46] may be generalized to apply here. (vii) So far, the implementation of the presented approach has been done without using a DEVSimPy plug-in. However, we have used an object-oriented programing approach and we have isolated all of the code in preparation for its integration into a specific DEVSimPy plug-in.

Funding

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

References

1. Zeigler BP and Sarjoughian HS. *Guide to modeling and simulation of systems of systems*. London: Springer-Verlag, 2013.
2. Zeigler BP. *Theory of modeling and simulation*. New York: Wiley, 1976.
3. Zeigler BP, Kim TG and Praehofer H. *Theory of modeling and simulation*. 2nd ed. Orlando, FL: Academic Press, Inc., 2000.
4. Zeigler BP and Sarjoughian HS. System entity structure basics. In: *Guide to modeling and simulation of systems of systems*, Simulation Foundations, Methods and Applications. London: Springer, pp.27–37.
5. Kim TG, Lee C, Christensen ER, et al. System entity structuring and model base management. *IEEE Trans Syst Man Cybern* 1990; 20(5): 1013–1024.
6. Rozenblit JW and Zeigler BP. Representing and constructing system specifications using the system entity structure concepts. In: *25th winter simulation conference*. *WSC '93*, Los Angeles, CA, 12–15 December 1993, pp.604–611. New York, NY: ACM.
7. Zeigler BP, Seo C and Kim D. System entity structures for suites of simulation models. *Int J Model Simula Sci Comput* 2013; 04(03): 1340006.
8. Zeigler BP, Seo C, Coop R, et al. Creating suites of models with system entity structure: Global warming example. In: *Symposium on theory of modeling & simulation*, San Diego, CA, 7–10 April 2013, pp.1–8. Society for Computer Simulation International.
9. Santucci JF, Capocchi L and Zeigler BP. SES extension to integrate abstraction hierarchy into DEVS modeling and simulation. In: *2015 spring simulation multiconference*, Alexandria, VA, 12–15 April 2015. Society for Computer Simulation International. pp.782–789.
10. Benjamin P, Erraguntla M, Delen D, et al. Simulation modeling at multiple levels of abstraction. In: *Simulation winter conference*, Washington DC, 13–16 December 1998, pp.391–398. IEEE Computer Society Press.
11. Fishwick PA and Lee K. Two methods for exploiting abstraction in systems. In: *Simulation and planning in high autonomous systems*, Tuscon, AZ, 1996, pp.257–264. IEEE Computer Society Press.
12. Zeigler BP, Praehofer H and Kim TG. *Theory of modeling and simulation*. 2nd ed. London: Academic Press, 2000.
13. Capocchi L, Santucci JF, Poggi B, et al. DEVSimPy: A collaborative Python software for modeling and simulation of DEVS systems. In: *20th IEEE international workshops on enabling technologies*, Paris, France, 27–29 June 2011, pp.170–175. IEEE Computer Society Press.
14. Van Tendeloo Y and Vangheluwe H. The modular architecture of the Python(P)DEVS simulation kernel (WIP). In: *Symposium on theory of modeling & simulation – DEVS integrative*, Tampa, Florida, USA, 13–16 April 2014, pp.14:1–14:6. San Diego, CA: Society for Computer Simulation International.
15. Van Mierlo S, Mustafiz S, Barocca B, et al. Explicit modelling of a parallel DEVS experimentation environment. In: *Symposium on theory of modeling & simulation*, Alexandria, VA, USA, 12–15 April 2015, pp.860–867. San Diego, CA: Society for Computer Simulation International.
16. Fard M and Sarjoughian H. *Visual and persistence behavior modeling for DEVS in CoSMoS*, volume 47. 8 ed. San Diego, CA: The Society for Modeling and Simulation International, 2015. pp.227–234.
17. Sarjoughian HS and Elamvazhuthi V. CoSMoS: A visual environment for component-based modeling, experimental design, and simulation. In: *2nd international conference on simulation tools and techniques*, Rome, Italy, 2–6 March 2009, pp.1–9. Brussels, Belgium: Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering. Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).
18. Muzy A, Capocchi L and Santucci JF. Using activity metrics for DEVS simulation profiling. *ITM web of conferences*,

Zuirch, Switzerland, 16–18 January 2014, Vol. 3, pp.1–9. Les Ulis, France: EDP Sciences - Web of Conferences.

19. Bettini C, Jajodia SG and Wang SX. *Time granularities in databases, data mining and temporal reasoning*. 1st ed. Secaucus, NJ: Springer-Verlag New York, Inc., 2000.
20. Montanari A. *Metric and layered temporal logic for time granularity*. PhD Thesis, University of Amsterdam, Netherlands, 1996.
21. Franceschet M and Montanari A. *Dividing and conquering the layered land*. PhD Thesis, University of Udine, Italy, 2001.
22. Euzenat J. Granularity in relational formalisms with application to time and space representation. *Comput Intell* 2001; 17(4): 703–737.
23. Guo S, Hu X and Wang X. On time granularity and event granularity in simulation service composition (WIP). In: *2012 symposium on theory of modeling and simulation – DEVS integrative M&S symposium*, Orlando, Florida, USA, 26–29 March 2012, pp.1–8. San Diego, CA: Society for Computer Simulation International.
24. Zeigler BP and Hammonds PE. *Modeling & simulation-based data engineering*. London: Academic Press, 2007.
25. Cheon S, Doohwan K and Zeigler BP. DEVS model composition by system entity structure. In: *IEEE international conference on information reuse and integration*, Las Vegas, Nevada, USA, 13–15 July 2008, pp.479–484. IEEE.
26. Tekinay C, Seck MD and Verbraeck A. Exploring multi-level model dynamics: Performance and accuracy (WIP). In: *2012 symposium on theory of modeling and simulation*, Orlanda, Florida, 26–29 March 2012, pp.1–6. San Diego, CA: Society for Computer Simulation International.
27. Steiniger A, Krüger F and Uhrmacher AM. Modeling agents and their environment in multi-level-DEVS. In: *Winter simulation conference*, Berlin, Germany, 9–12 December 2012, pp.233:1–233:12. New York: IEEE.
28. Himmelspach J and Uhrmacher A. The JAMES II framework for modeling and simulation. In: *International workshop on high performance computational systems biology*, Trento, Italy, 14–16 October 2009, pp.101–102. New York: IEEE.
29. Mittal S. Emergence in stigmergic and complex adaptive systems: A formal discrete event systems perspective. *Cognit Syst Res* 2013; 21(0): 22–39.
30. Seck MD and Honig HJ. Multi-perspective modelling of complex phenomena. *Comput & Math Organiz Theory* 2012; 18(1): 128–144.
31. Davis P and Bigelow J. Experiments in multi resolution modeling (MRM). RAND Report MR1004, RAND Corp, 1998.
32. Moon IC and Hong JH. Theoretic interplay between abstraction, resolution, and fidelity in model information. In: *Winter simulation conference*, Washinton DC, USA, 8–11 December 2013, pp.1283–1291. New York: IEEE.
33. Baohong L and Kedi H. A formal description specification for multi-resolution modeling (MRM) based on DEVS formalism. In: *13th international conference on AI, simulation, and planning in high autonomy systems*, Jeju Island, Korea, 4–6 October 2004, pp.285–294. Berlin, Heidelberg: Springer-Verlag.
34. Barros FJ. Modeling formalisms for dynamic structure systems. *ACM Trans Model Comput Simul* 1997; 7(4): 501–515.
35. Aïello A, Santucci JF, Delhom M, et al. Modélisation multi-vue et simulation à événements discrets. *L'Objet* 2000; 6(4).
36. Wainer GA. *Discrete-event modeling and simulation: A practitioner's approach*. 1st ed. Boca Raton, FL: CRC Press, Inc., 2009.
37. Seck MD and Honig HJ. Multi-perspective modelling of complex phenomena. *Comput Math Organ Theory* 2012; 18: 128–144.
38. Hu X, Zeigler BP and Mittal S. Variable structure in DEVS component-based modeling and simulation. *Simulation* 2005; 81(2): 91–102.
39. Tendeloo YV. *Activity-aware DEVS simulation*. Master's Thesis, University of Antwerp, 2014.
40. Coron L, Andréassian V, Perrin C, et al. On the lack of robustness of hydrologic models regarding water balance simulation: A diagnostic approach applied to three models of increasing complexity on 20 mountainous catchments. *Hydrol Earth Syst Sci* 2014; 18(2): 727–746.
41. Valéry A, Andréassian V and Perrin C. Regionalization of precipitation and air temperature over high-altitude catchments – Learning from outliers. *Hydrol Sci J* 2010; 55(6): 928–940.
42. Taylor R. Interpretation of the correlation coefficient: A basic review. *J Diagn Med Sonography* 1990; 6(1): 35–39.
43. Seo C, Zeigler BP, Coop R, et al. DEVS modeling and simulation methodology with MS4Me software tool. In: *Symposium on theory of modeling & simulation*, San Diego, CA, USA, 7–10 April 2013, pp.1–7. San Diego: Society for Computer Simulation International.
44. Sarjoughian HS and Sundaramoorthi S. Superdense time trajectories for DEVS simulation models. In: *Symposium on theory of modeling & simulation*, Alexandria, VA, USA, 12–15 April 2015, pp.249–256. San Diego, CA: Society for Computer Simulation International.
45. Zeigler BP, Nutaro JJ and Seo C. Combining DEVS and model-checking: Concepts and tools for integrating simulation and analysis 2016; To appear in Int. J. Process Modeling and Simulation. *I3M* 2014: Special Issue on: ''New Advances in Simulation and Process Modelling: Integrating New Technologies and Methodologies to Enlarge Simulation Capabilities.
46. Pawletta T, Schmidt A, Zeigler BP, et al. Extended variability modeling using system entity structure ontology within MATLAB/Simulink. In: *SCS international SpringSim/ANSS*, Pasadena, CA, USA, 3–6 April 2016, pp.62–69. San Diego, CA: Society for Computer Simulation International.