Reusability Measure of DEVS Simulation Models in DEVSim++ Environment

Yoonil Choi and Tag Gon Kim

Department of Electrical Engineering
Korea Advanced Institute of Science and Technology
373-1 Kusong-dong Yusong-gu Taejon 305-701, Korea

ABSTRACT

This paper deals with reusability of the DEVS (Discrete Event Systems Specification) models in the hierarchical models development framework within an object-oriented simulation environment, called DEVSim++. The DEVSim++ environment supports models reusability in two dimensions during models development. One way of reusability is achieved from the hierarchical model construction technology from the DEVS formalism and the other from the inheritance mechanism from the underlying object-oriented environment. This paper proposes a set of metrics to measure both hierarchical reuse and inheritance reuse of DEVS models developed in DEVSim++. It also suggests a set of guidelines to improve reusability. Empirical measurement of the proposed metrics shows that the guidelines improve reusability of DEVS simulation models in the DEVSim++ environment.

Keywords: Models Reusability, Reusability metrics, DEVS formalism, DEVSim++, Object-oriented Simulation

1. Introduction

Software reuse is an important technology in software engineering field, especially in object-oriented software engineering.[1,2] The technology seeks to increase productivity and quality of software systems by reducing cost and effort in software development. Although modeling/simulation is a software engineering process no research has been conducted in reusability for discrete event simulation models.

The DEVS formalism, developed by Zeigler, supports specification of discrete event models in hierarchical, modular manner.[3] And DEVSim++ is an object-oriented simulation environment which realizes the DEVS formalism in C++.[4] It has been noticed that the DEVSim++ environment supports models reusability in two dimensions during models development.[5] One way of reusability is achieved from the hierarchical model construction technology from the DEVS formalism and the other from the inheritance mechanism from the object-oriented DEVSim++ environment.

Quantitative study for reusability measure is required to monitor and improve the reuse process in simulation models development. Such study gives modelers important guidelines for models reuse. This paper deals with reuse of the simulation models in the models development process using DEVS models within the DEVSim++ environment. This paper proposes a set of metrics to measure reusability of simulation models. The metrics can measure both hierarchical reuse and inheritance reuse of DEVS models in DEVSim++. It develops a tool for empirically measure such metrics automatically from the source codes. A set of guidelines to improve reusability is suggested. Empirical measurement of the proposed metrics shows that the guidelines improve reusability of DEVS simulation models in the DEVSim++ environment.

This paper is organized as follows. Section 2 reviews the DEVS formalism and the DEVSim++ environment. Section 3 presents a framework for models reusability. Section 4 derives reusability metrics, and introduces measuring tools. Section 5 shows empirical measurements for the proposed metrics. The reuse guidelines are suggested in section 6, and section 7 concludes the paper.

Further author information -
Y.C.: Email:yichoi@core.kaist.ac.kr; Telephone: +82-42-869-8054; FAX: +82-42-869-8025
T.G.K.: Email:tkim@ee.kaist.ac.kr; Telephone: +82-42-869-3454 FAX: +82-42-869-8025

SPIE Vol. 3083 • 0277-786X/97/$10.00

2. The DEVS Formalism and DEVSim++

This section briefly reviews the DEVS formalism and DEVSim++, an implementation of the formalism in a C++ environment.

2.1. DEVS Formalism

A set-theoretic formalism, the DEVS formalism specifies discrete event models in a hierarchical, modular form. Within the formalism, one must specify 1) the basic models from which larger ones are built, and 2) how these models are connected together in hierarchical fashion.[7] A basic model, called an atomic model (or atomic DEVS), has specification for dynamics of the model as:

$$AM = < X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta >$$

X : input events set;
S : sequential states set;
Y : output events set;
δ_{int} : $S \rightarrow S$: internal transition function;
δ_{ext} : $Q \times X \rightarrow S$: external transition function;
$\lambda : S \rightarrow Y$: output function;
$ta : S \rightarrow Real$: time advance function,

where $Q = \{(s, e) \mid s \in S, 0 \leq e \leq ta(s)\}$: total state of AM

An atomic model specifies the behavior of a discrete event system as a timed state transition machine. The set S includes all possible states of the system and $ta(s)$ maps s to a non-negative real with infinity, which is the time during which the system is allowed to stay in state s if no external event occurs. Atomic DEVS model employs two types of state transitions, internal transition and external transition. The internal transition function δ_{int} defines the next state when no external event occurs at a current state. If an external event occurs before completion of an elapsed time for a current state, the next state of the system is determined by the external transition function δ_{ext}. We call the four functions in the 7-tuple, namely, internal transition function, external transition function, time advance function and output function, the DEVS characteristic functions.

The second form of the model, called a coupled model (or coupled DEVS), defines how to couple (connect) several component models together to form a composite model. This latter model can itself be employed as a component in a larger coupled model, thus giving rise to construction of complex models in hierarchical fashion. The definition of a coupled model CM is as follows.

$$CM = < X, Y, M, EIC, EOC, IC, SELECT >$$

X : input events set;
Y : output events set;
M : DEVS components set;
$EIC \subseteq CM.IN \times M.IN$: external input coupling relation;
$EOC \subseteq M.OUT \times CM.OUT$: external output coupling relation;
$IC \subseteq M.OUT \times M.IN$: internal coupling relation;
$SELECT : 2^M - \emptyset \rightarrow M$: tie-breaking selector,

where the extensions .IN and .OUT represent the input ports set and output ports set of respective DEVS models.

The three coupling relations, namely, EIC, EOC, and IC, specifies input/output connections between component models and the coupled model. The coupling scheme (CS) of a coupled model is defined as a collection of the three relations, (EIC, EOC, IC). The coupling scheme plays the main role in hierarchical construction of DEVS models.Detailed descriptions for the definitions of the atomic and coupled DEVS can be found in Ref. 7.

2.2. The DEVSim++ Environment

The DEVSim++ environment is a realization of the DEVS formalism in C++. It has been developed and upgraded in CORE laboratory in KAIST since 1992.[4] It supports modelers to develop discrete event models using the hierarchical composition methodology in an object-oriented framework. The environment is a result of the combination of two powerful frameworks for systems development : the DEVS formalism and the object-oriented paradigm.

The DEVSim++ environment supports class definitions for modelers to develop atomic and coupled DEVS models. The definitions are the same as ones defined in the DEVS formalism for atomic and coupled models. Modelers using the DEVSim++ environment can develop DEVS models within a C++ environment, thus exploiting the hierarchical modeling methodology from the DEVS formalism and the inheritance mechanism from the object-oriented environment. Therefore, the DEVSim++ environment supports models reusability in two dimensions during models development: hierarchical composition based reuse and inheritance based reuse. To exploit such reusability effectively is modeler's responsibility. For example, to exploit the inheritance based reuse a modeler may define subclasses of atomic or coupled model class defined in DEVSim++ during the models development process. The class hierarchy for the DEVSim++ environment is shown in Figure1.

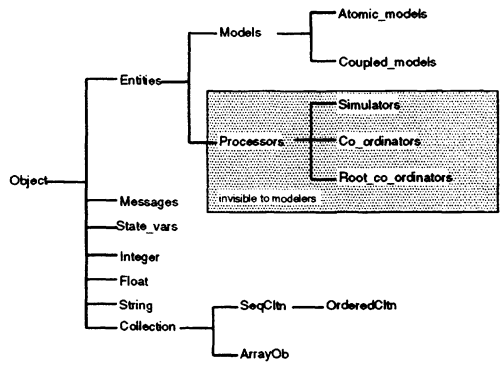

Figure 1. DEVSim++ Class Hierarchy

Detail description for DEVSim++ is available in Ref 8.

3. Framework for Model Reuse

There are two software reuse technologies. One is compositional reuse technology and the other is generational reuse technology.[1] In composition reuse technology, components such as code skeletons, subroutines, functions, modules, classes and programs are reused as building blocks in developing a software. In generation reuse technology, patterns are reused to generate a software. Compiler is one of generation reuse technology examples.

Simulation model reuse is more composition technology than generation technology. In other words, software artifacts are usually reused as components in simulation model development. The following five reuse rules were suggested for model reuse in Ref. 5.

R1. Functions may be reused in development of models.

R2. Data may be reused in development of models.

R3. Models may be reused in construction of composite models which in turn are reused as components of higher-level composite models.

R4. Models may be reused in development of new models that are slightly different from old ones.

R5. Models may be reused in a variety of applications.

The principle applied in R1 is function abstraction. It is widely used in specification of transition functions of a discrete event system, which represents dynamics of the system in terms of a sequence of state transitions. Functions so defined may be reused by applying different arguments in the required type to them. The principle applied in R2 is data abstraction. In data abstraction, data are reused by a family of admissible functions applied to it. In the modeling context, a set of state variables is an example of such data.

The principle applied to R3 is the closure under coupling property in the DEVS formalism. In hierarchical model composition process within the DEVS formalism, models built from aggregating lower level models can be reused as component models in higher level model composition. Figure 2 shows an example of hierarchical model composition.

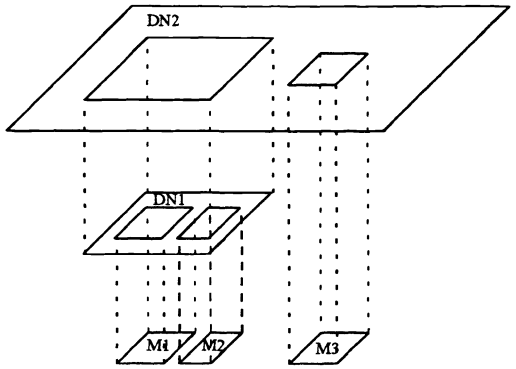

Figure 2. Example of Hierarchical Model Composition

M1, M2, M3 are atomic models and DN1, DN2 are coupled models. In developing DN2, M1 and M2 are used as components. In turn, DN1 and M3 are used as components for DN2. Since DN1 has interfaces in the same way of atomic models, DN1 can be used as a component in developing DN2. This is the closure under coupling principle. For more detailed explanation of hierarchical model composition, refer to Ref. 3.

R4 is modification reuse. It is more often the case to modify existing models to reuse than to reuse models as they are. The object-oriented paradigm supports modification reuse through inheritance which is the disciplined modification reuse methodology. Thus, DEVSim++ supports inheritance to reuse existing models. Modification reuse through inheritance is a major reuse framework with hierarchical model composition in simulation modeling. Reusability exploiting both inheritance and hierarchical model composition are called two dimensional reusability. Figure 3 shows the two dimensional reusability.

R5 is related to how frequently models are reused in applications. R5 is an issue of the library technology which we shall not discuss here. We base our reusability framework on the composition technologies of R1 to R4. The framework was named as the object-oriented, hierarchical composition methodology in Ref. 5.

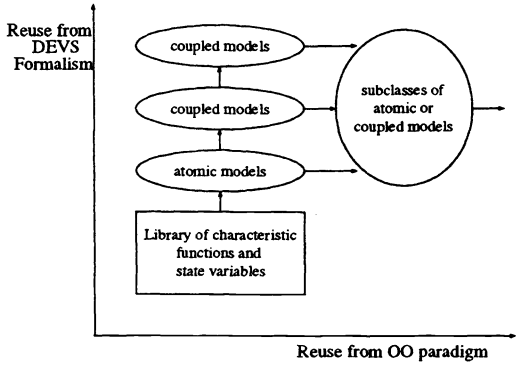

Figure 3. 2-Dimensional Reusability in DEVSim++

4. Reusability Metrics

This section proposes the metrics for the reusability of DEVS models in DEVSim++ after introducing a framework to derive metrics in software measure.

4.1. Framework for Deriving Metrics

The measure theory provides a formal framework to define meaningful metrics. Following four steps are the framework for deriving metrics.[9]

1. Identify and define intuitive and well-understood attributes of software to be measured, and specify precisely the documents and the attributes to be measured. We must be able to describe precisely the object that we measure, and the property of the object that the measurement is to indicate.
2. Determine formal models or abstractions which capture these attributes. Formal models are required to unambiguously produce numerical measurement values.
3. Identify important relationships and orderings that exist between the objects(being modeled) and that are determined by the attributes of models.
4. Derive mappings from the models of attributes to a number system. The mappings must be consistent with the orderings implied by the model attributes.

The following subsections derive the metrics for reusability in DEVS simulation modeling according to the formal framework for deriving metrics from the measure theory. First, the reuse attributes are specified, and then the abstractions which represent the attributes are defined. Finally, the reusability metrics are defined from the abstractions to quantify the attributes.

4.2. Reuse Attributes

Based on the identified simulation model reusability in Ref. 5, the reuse attributes can be specified in four categories - hierarchical model composition, inheritance, function abstraction and data abstraction. Function abstraction and data abstraction are applied in the course of hierarchical model composition. Thus, we specify compositional reuse through hierarchical models construction and modification reuse through inheritance.

Followings are the reuse attributes of hierarchical composition.

H1. A model can be reused in developing composite models

H2. A composite model includes previously developed models as components

H3. A composite model can be reused in developing higher-level models

The next three reuse attributes are of modification reuse.

I1. Modification Reuse takes place through inheritance by adding state variables*, adding methods and overloading methods

I2. A class can inherit from more than one class

I3. A inherited class can have children classes inherited from it

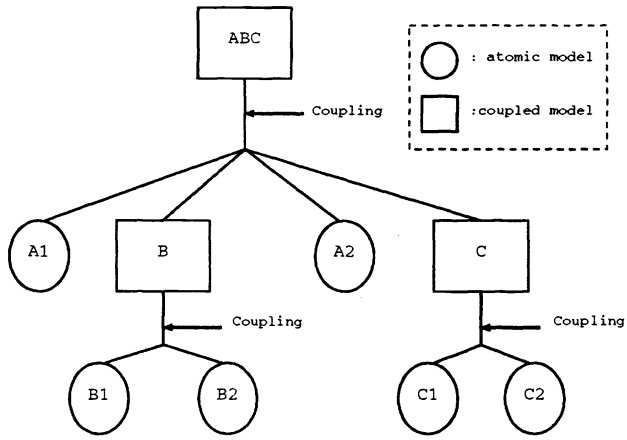

Figure 4. Composition Tree

4.3. Abstractions

Now, this subsection derives the reuse abstractions based on the reuse attributes defined above. Note that two major principles of model reuse are hierarchical model composition from DEVS formalism and modification reuse through inheritance from object-oriented paradigm. Therefore, two abstractions for each principle are suggested.

We suggest *composition tree* as the abstraction for hierarchical model composition. A composition tree exhibits the structure of a hierarchical model.[10] Figure 4 shows a composition tree example. The root model ABC is the outermost coupled model and its children are the children nodes of the tree(e.g., A1, B, A2 and C). A child B of a coupled model ABC has also its own children B1 and B2. Atomic models A1, A2, B1, B2, C1 and C2 become leaf nodes of the composition tree. Basically, a composition tree is a kind of tree, so it has every property that a tree has. The properties includes dimensions such as number of nodes, level of a node, and depth of a tree, which have partial ordering.

For modification reuse through inheritance, *class inheritance hierarchy graph* is suggested as the abstraction. A class inheritance hierarchy graph is the same as a tree except that a child node can have more than one parent node. A class inheritance hierarchy graph includes every property of a composition tree. Every dimension such as number of nodes, level of a node, and depth of a tree is defined similarly to those of a tree with some conditions. For instance, the level of a node is the maximum length from the node to the root of the tree in the case the node has more than one parent. In this context, a class inheritance hierarchy graph has the same partial ordering as a composition tree. Figure 5 shows an example of a class inheritance hierarchy graph. With these two abstractions, the reusability metrics will be defined in the next subsection by finding mappings from the abstractions to a real number system.

4.4. Reusability Metrics for DEVS Simulation Models

Based on the framework for defining metrics, this section defines reusability metrics from the two abstractions suggested in the previous section. Six metrics to measure models reusability are defined. Three derives from the composition tree of DEVS models, which reflects hierarchical composition based reuse inherited from the DEVS formalism. The other three are adopted from Ref. citenumChidamber94 with some variation, which reflects modification reuse from the object-oriented paradigm.

Metric 1 : NRM (Number of Reuse of a Model in building higher models)

(1) Definition : number of identical nodes of a model in a composition tree
(2) View points
- NRM reflects the reuse attribute, H1.

*The word "state" here is different from the one in DEVS models though the "state" is named from the same concept.

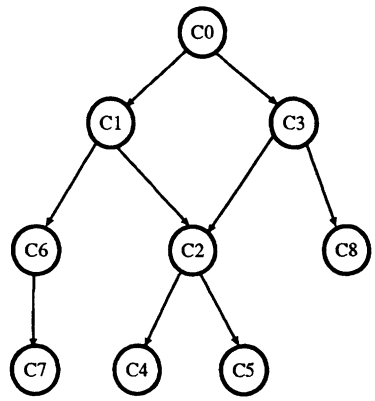

Figure 5. Class Inheritance Hierarchy Graph

- NRM is the most intuitive reusability metric.
- NRM shows how many times a model is included in constructing higher composite models.

Metric 2 : NCM (Number of Component Models in a coupled model)

(1) Definition : number of component model nodes of a coupled model node in a composition tree.
(2) View points
- NCM reflects the reuse attribute, H2.
- NCM measures how many component models are reused in developing a coupled model.

Metric 3 : DCT (Depth of a Composition Tree)

(1) Definition : Depth of a composition tree
(2) View points
- DCT reflects the reuse attributes, H3.
- DCT shows the reusability from the closure under coupling principles. A model which is developed from the composition of component models can be used as a component model in developing higher level models. DCT shows how many times this principle is applied to develop the final model.

Metric 4 : NOC (Number of Children classes)

(1) Definition : number of immediate subclasses subordinated to a non-leaf class in the class hierarchy.
(2) View points
- NOC reflects the reuse attribute, I1.
- Measures how many times a class is reused through inheritance
- Children use their parent's state variables and methods.

Metric 5 : NOP (Number of Parents of a class)

(1) Definition : number of parent classes which a class is inherited from.
(2) View points
- NOC reflects the reuse attribute, I2.
- NOP measures multiple inheritance.

Metric 6 : DCIHG (Depth of a Class Inheritance Hierarchy Graph)

(1) Definition : maximum path length from the root class node.
(2) View points
- DCIHG reflects the reuse attribute, I3.
- The deeper a particular class is in the hierarchy, the greater the potential reuse of inherited methods and state variables.

4.5. Measuring Tools

The automatic measuring tools are required to measure the metrics defined in the previous section directly from the DEVSim++ source code. We developed two tools - *composition reuse measurer and modification reuse measurer.*

The *composition reuse measurer* measures the reusability in hierarchical model composition - DCT, NCM, and NRM. The *modification reuse measurer* is for metrics which are the mappings from the class inheritance hierarchy graph to a real number system - NOC, NOP, and DCHIG.

Figure 6 shows the basic idea of the measuring process which is the main algorithm of the two measuring tools. Since the reuse metrics are from the abstractions, the tools first extract the abstractions from DEVSim++ source

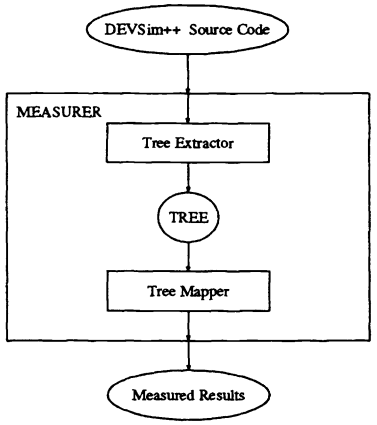

Figure 6. Measuring Process

codes by means of lexical analysis. Then, according to the definition of each metric, the tools count numbers from the abstractions. This process results in the values of each metric.

5. Empirical Results

Empirical data and their analysis are given in this section. Empirical data are obtained from measuring 23 DEVSim++ models developed in KAIST. Complete set of empirical data is given in Table 1 and 2.

The median DCT of the given models was 2.5, which is remarkably small. The statistic tells that the compositional reuse weren't much exploited through the closure under coupling principle, and that the models didn't reuse any existing models but were built from scratch. As a matter of fact, all of the models measured here don't contain any previously developed models.

Most coupled model have about 4 component models. NCM depends on the structure of models. That is, the models which have the repetitive nature may have large NCM's. In this respect, the models of M5, M8, and M11 can be thought that they may contain homogeneous repetitive components because those models have remarkably large NCM's. M9 also has large median NCM values. In fact, M5 is a hypercube parallel computer model which contains identical node computers. M8 is the model of an intelligent vehicle highway system in which the highway lane models are highly reused, M9 contains the identical node models of the control/data flow graph, and M12 is the traffic control model also containing the identical lane models.

The median NRM values of a half the models equals to 1. This result means that the model classes have only one instance, thus being lack of the model reuse clearly. We can infer there are two reasons for that. One is that the models are so simple that the number of components in a model would be too small; the other is that models have been ill-abstracted. In the second case, we can improve the model reuse by controlling the model abstraction levels.

It is found that there are some models reused often as components. Especially, Buffer(or queue) models are frequently reused in the different forms. M7 has 160 buffer models, and M5, M12 and M14 have 20 or more models.

Model	DCT	NCM				NRM			
		min	median	max	mean	min	median	max	mean
M1	2	3	3	6	3.833	1	4	4	2.875
M2	5	2	3	13	4.190	1	4	16	4.407
M3	2	2	2	7	2.625	1	2	4	2.333
M4	1	5	5	5	5.000	5	5	5	5.000
M5	3	2	3	32	6.268	32	32	192	67.556
M6	3	2	3	9	2.833	1	8	8	5.667
M7	4	1	3	16	5.392	1	16	80	22.917
M8	3	4	10	39	9.000	1	10	40	15.500
M9	2	3	13	13	8.000	1	1	11	3.200
M10	2	2	2	2	2.000	1	1	1	1.000
M11	3	2	10	41	12.429	1	1	20	2.900
M12	2	2	2	5	3.000	1	1	1	1.000
M13	2	4	5	5	4.500	1	1	1	1.000
M14	2	2	3	16	3.500	1	5	20	8.000
M15	3	2	5	8	4.636	1	4	12	4.636
M16	4	1	2	7	2.571	1	1	2	1.125
M17	3	2	2	9	2.727	1	2	8	3.750
M18	4	2	2	3	2.182	1	1	2	1.143
M19	4	1	2	10	2.600	1	1	10	3.250
M20	2	2	2	11	3.167	1	1	1	1.000
M21	2	2	3	9	3.500	1	1	5	1.750
M22	2	2	2	4	2.667	1	1	1	1.000
M23	1	14	14	14	14.000	1	1	1	1.000

Table 1. Measuring Results of Composition Tree Metrics

It would have be better that a general buffer(or queue) model had been in a repository system such as library to reuse in developing those models.

Table 2. shows metrics for class inheritance hierarchy. In the table, the DCIHG's of all the models are 3 or 4. The result tells that inheritance didn't occur much and model modification reuse has never been exploited. The result also shows that average NOC of non-leaf classes have 2 or 3. That means non-leaf classes are reused 2 or 3 times on average.

NOP's of all classes were found to be 1[†]. The results implies that there was no multiple inheritance.

6. Guideline

We found that the empirical models shown in section 4 have been built from scratch, not from reusing existing models as components. It is because there were not any methodologies to reuse existing models such as a repository system such as model base[10‡]. It was found that some empirical models were ill-abstracted and didn't have their inheriting models. That is, another important factor for reuse in simulation modeling is the model's abstraction level. The more generic, the better reusable. Therefore, it is necessary to develop models with abstraction hierarchy

[†]root classes have no parent classes.

[‡]A model base, which exploits the principle of closure under coupling in DEVS theory, is an organized library of such reusable models. Models in a model base can be either atomic or coupled ones. New models can be saved in, and saved models can be retrieved from, the model base. Models so retrieved may be reused to construct a hierarchical model which can be reused as a component model in construction of yet more complex, hierarchical models. Again, the complex, hierarchical model is saved in the model base and can be reused later, and so on.

Model	DCHIG	NOC				NOP
		min	median	max	mean	max
M1	4	1	3	7	3.800	1
M2	4	1	2	21	7.000	1
M3	4	1	2	6	2.714	1
M4	4	1	1	6	2.200	1
M5	3	1	2	9	4.000	1
M6	3	1	1.0	15	3.500	1
M7	3	1	1	13	3.600	1
M8	3	1	1	10	3.000	1
M9	3	1	1	6	2.200	1
M10	3	1	2	6	3.000	1
M11	3	1	2	5	2.667	1
M12	3	1	1	6	2.200	1
M13	3	1	2	6	3.000	1
M14	3	1	1.5	7	2.333	1
M15	3	1	2	13	3.800	1
M16	3	1	1	7	2.400	1
M17	3	1	2	7	2.600	1
M18	3	1	1	14	3.800	1
M19	3	1	1	7	2.400	1
M20	3	1	1	6	2.200	1
M21	3	1	1	6	2.200	1
M22	3	1	1	6	2.200	1
M23	3	1	2	5	2.670	1

Table 2. Measuring Results of Class Inheritance Hierarchy Graph Metrics

to improve reusability. However, it mainly depends on modelers' ability[§]. Miscellaneously, it was found that the empirical models have no modification model reuse and no multiple inheritance.

Based on the empirical data and observation, we suggest the following guidelines to improve reusability of simulation models. Figure 7 shows the main idea of the guideline.

Step 1. Before modeling a system, examine it carefully to find reusable attributes which were presented in section 3. Find repetitive nature(or component) of the system to be modeled.

Step 2. Considering the attributes found in the first step, draw a composition tree with abstraction hierarchy in the way of decomposing a entire model into several component models, in turn, decomposing each component model into one's own component models.

Step 3. Examine behavioral similarity between models in the composition tree, There may be component models with similar behavior between the component models. Find those models and classify them.

Step 4. Draw a class inheritance hierarchy graph. In this step, The reusable attributes found in the first step should be considered again. In the case of existence of a repository system, find as many necessary models as possible from a repository system. Reusing a repository system such as model base is the most desirable way of reuse since the models in the library can be thought of as well-maintain, quality models.

Step 5. Develop abstract models first. There may be so similar components in system models that they can be built from modifying the abstract(or generic) models. If there is a repository system, the abstract models can be extracted from it, and the developed abstract models can be stored in it.

[§]Modelers ability can be improved through training, i.e., education.

Model	NRM				DCIHG	NOC			
	min	median	max	mean		min	median	max	mean
M9	1	1	11	3.2	3	1	1	6	2.2
M9'	1	1	13	4	4	1	2	6	2.286
M12	1	1	1	1	3	1	1	6	2.2
M12'	1	1	2	1.125	4	1	2	6	2.857
M13	1	1	1	1	3	1	2	6	3
M13'	1	1	2	1.286	4	1	2	6	3.2

Table 3. Reuse Metric Values of Original and Modified Models

Step 6. Then, Develop specific models by modifying abstract models through inheritance. DEVSim++ has powerful reuse framework supported by object-oriented paradigm - inheritance. So, modelers can develop models with abstract hierarchy which increases reuse. If there is a repository system, the specific models can be extracted from it, and the developed specific models can be stored in it.

According to the guideline, the three empirical models, M9, M12, and M13 are modified and the reusability of the modified models are measured. Table 3 shows the improved values of the reuse metrics. M9', M12' and M13' in the Table 3 are the modified models from M9, M12, and M13, respectively.

To get a view that the guideline work in the expected way, the uncommented lines of codes for original models and their modified models are measured. Table 4 shows the uncommented lines of codes. Note that every modified model has reduced LOC's comparing the original. These results can tell that the proposed reuse guideline takes reuse advantages - productivity, maintainability and so on.

Table 4 also shows performance in simulation execution time for original and modified models to follow reuse guidelines. The simulation times of the modified models are little more than those of the original models. The simulation overhead in modified models are 0.32 to 1.44 %. It mostly results from the memory reference overhead. Since the class hierarchy of every modified model is deeper than the original one, the memory references of the modified one take more time.

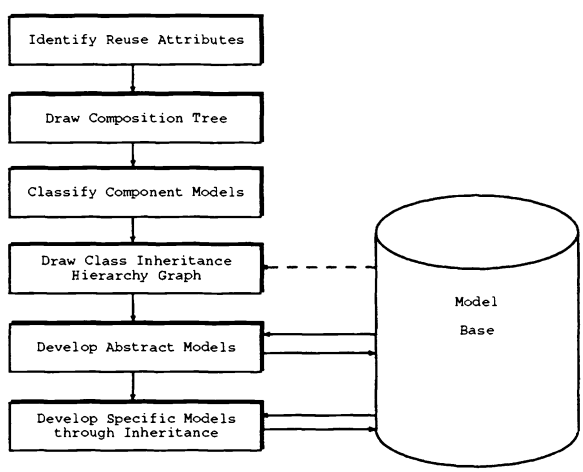

Figure 7. Guideline to Improve Reusability for Simulation Models

Model	# Atomic Models	# Lines			Execution Time (s)		
		original	modified	% reduced	original	modified	% overhead
M9	15	808	737	8.80 %	528.5	535.2	1.27 %
M12	7	1001	948	5.29 %	186.9	187.5	0.32 %
M13	8	1106	909	17.81 %	367.7	373.0	1.44 %

Table 4. Lines of Codes for Original Models and Their Modification

7. Conclusion

The main purpose of this paper is to define and measure reusability DEVS models developed in the DEVSim++ environment, thereby find the ways to improve the reusability. For the first purpose, we defined the 6 reuse metrics from the reuse attributes of DEVS models in DEVSim++, and developed tools to measure such metrics empirically. For the second purpose, we suggested a guideline to improve reusability of simulation models based on empirical data obtained from DEVSim++ simulation models. In simulation modeling, it is recommended to find repetitive nature (or component) of the system to be modeled, classify the component models. We develop generic abstract models first, and then develop specific models by modifying abstract models through inheritance. In the case of existence of a repository system, it is also recommended to reuse models in the repository system and store newly-developed models into it.

Ultimately, the model reuse engineering should first be directed to develop a model repository system, and maintain and update it. It then devises a methodology for reusable models design with a repository system. Finally, a computer aided modeling system based on such methodology needs to be developed.

REFERENCES

1. T. Biggerstaff and C. Richter, "Reusability framework, assessment, and directions," in *Software Reusability*, T. Biggerstaff and A. J. Perlis, eds., pp. 1–41, ACM Press, New York, 1989
2. J. Bieman, "Deriving measures of software reuse in object-oriented systems," in *Formal Aspects of Measurement*, T. Denvir, R. Herman, and R. Whitty, eds., *Workshop on Formal Aspects of Measurement* , pp. 79–82, 1992.
3. . A. Concepcion and B. Zeigler, "DEVS formalism: A framework for hierarchical model development," *IEEE Trans. on Software Engineering* **14**, pp. 228–241, 1994.
4. T. Kim and S. Park, "The DEVS formalism: Hierarchical modular systems specification in C++," *Proc. in European Simulation Multiconference* , 1992.
5. T. Kim and M. Ahn, "Reusable simulation models in an object-oriented framework," in *Object-oriented Simulation*, G. Zobrist, ed., IEEE Press, 1995.
6. T. Kim, "The DEVS formalism: Reusable model specification in an object-oriented framework," *Int. Journal in Computer Simulation* **5**, pp. 397–415, 1995.
7. B. Zeigler, *Multifacetted Modeling and Discrete Event Simulation*, Academic Press, Orlando, FL, 1984.
8. T. Kim, *DEVSim++ User's Manual*, ftp(anonymous)://sim.kaist.ac.kr/pub/devsimg++.tar.Z, Department of Electrical Engineering, KAIST, 1994.
9. A.L. Baker, J.M. Bieman, N.E. Fenton, A.C. Melton, "A philosophy for software measurement," *Journal of Systems and Software* **12**, pp. 277–281, 1990.
10. B. Zeigler, *Object-oriented Simulation with Hierarchical Modular Models*, Academic Press, Boston, MA, 1990.
11. S. Chidamber and C. Kemerer, "A metrics suite for object oriented design," *IEEE Trans. on Software Engineering* **20**, pp. 476–493, 1994.