Structure-Behavior Coalescence Abstract State Machine for Metamodel-Based Language in Model-Driven Engineering

Wei-Ming Ma, *Member, IEEE*, and William S. Chao

***Abstract*—In model-driven engineering (MDE), the unified modeling language (UML) 2.0 metamodel solution includes a metamodel that defines the language concepts and a user model that defines how the language concepts are represented. In UML 2.0, an important usage of metamodel is to ensure model consistency between different diagrams in the user model. However, most existing UML metamodels lack an integrated semantic framework to project each diagram in the user model as a view of the metamodel. To overcome the shortcomings of the current UML 2.0 metamodel approaches, we developed structure-behavior coalescence abstract state machine (SBC-ASM) for metamodel-based language (MBL), which provides an integrated semantic framework that is able to integrate structural constructs with behavioral constructs. Using SBC-ASM MBL as the metamodel solution of UML 2.0, each diagram in the user model can be projected as a view of the SBC-ASM.**

***Index Terms*—Abstract state machine (ASM), metamodel, metamodel-based language (MBL), model-driven engineering (MDE), structure-behavior coalescence (SBC), unified modeling language (UML), user model.**

I. Introduction

As a software modeling language for model-driven engineering (MDE) applications [1], the unified modeling language (UML) 2.0 profile approach [2], [3] defines a set of language concepts that are used to model the structure and behavior of a software system. The UML 2.0 concepts include 1) an abstract syntax that defines the language concepts and is described by a metamodel, and 2) a concrete syntax, or notation, that defines how the language concepts are represented and is described by a user model [4], [5].

UML 2.0 uses two types of diagrams to represent the user model: 1) Structure types include object diagrams, class diagrams, deployment diagrams, package diagrams, composite structure diagrams, component diagrams, etc. 2) Behavior types include use case diagrams, activity diagrams, state diagrams, sequence diagrams, communication diagrams, interaction overview diagrams, timing diagrams, etc. Since UML 2.0 is a multidiagram approach, there are always some inconsistencies between different diagrams in the user model [6]–[8]. Two obvious inconsistencies occur most often. The first inconsistency is between the structure type diagram and the behavior type diagram. For example, if the object described in sequence diagrams does not correspond to an instance of a normal class in class diagrams, there is an inconsistency. The second inconsistency is between two different behavior type diagrams. For example, if the ordered collection of stimuli received by an object in a sequence diagram does not exist as a sequence of events in the state diagram of that class, there is an inconsistency.

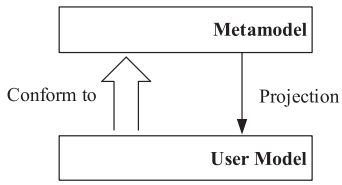

Fig. 1. UML 2.0 metamodel solution.

The metamodel with a single diagram is used to ensure model consistency between multiple diagrams in the user model. There is a model inconsistency if the user model does not conform to the metaclass properties, constraints, and relationship is defined by the metamodel. This can be done by defining a set of transformation (projection) rules as shown in Fig. 1. The conformation will be verified if all structural and behavioral diagrams in the user model can be transformed (projected) as a view of the metamodel.

Unfortunately, most current UML 2.0 metamodels are not able to project each diagram in the user model as a view of the metamodel. In this article, we develop structure-behavior coalescence abstract state machine (SBC-ASM) metamodel-based language (MBL) [9] as a metamodel solution of UML 2.0. In SBC-ASM MBL, each diagram in the user model will be projected as a view of the SBC-ASM metamodel. Therefore, we claim that SBC-ASM MBL genuinely provides a metamodel solution to ensure model consistency for UML 2.0.

The remainder of this article is arranged as follows. Section II deals with the current UML 2.0 metamodel study. SBC-ASM, MBL as a metamodel solution for UML 2.0 is detailed in Sections III and IV. Section V demonstrates a case study. After that, we will in Section VI verify the SBC metamodel solution through effort analysis. The conclusion of this article is in Section VII.

Manuscript received March 26, 2020; revised July 26, 2020 and September 19, 2020; accepted September 21, 2020. Date of publication October 13, 2020; date of current version August 26, 2021. *(Corresponding author: Wei-ming Ma.)*

Wei-Ming Ma is with the Information Management Department, Cheng Shiu University, Kaohsiung 83347, Taiwan (e-mail: k3666@gcloud.csu.edu.tw).

William S. Chao is with SBC Architecture International, Kaohsiung 80424, Taiwan (e-mail: architectchao@gmail.com).

Digital Object Identifier 10.1109/JSYST.2020.3027195

II. Literature Review

In UML 2.0, a metamodel is used to describe the concepts in the language, their characteristics, and interrelationships. This is sometimes called the abstract syntax of the language, and is distinct from the concrete syntax that specifies the user model for the language. A significant usage of the metamodel is to ensure model consistency between different diagrams in the user model.

The object management group defines a language for representing metamodels, called metaobject facility (MOF) as shown in Fig. 4 that is used for UML, Systems Modeling Language (SysML), and other metamodels. Several metamodeling mechanisms are used in MOF, such as object constraint language (OCL) [10], Foundational UML (fUML) [11], The Action Language for Foundational UML (Alf) [12], process specification language (PSL) [13], to name a few.

OCL is a precise text language that specifies structural semantics through invariants and behavioral semantics through the pre- and post- conditions on operations, but does not allow to alter the state of a model. Therefore, OCL is not sufficient for expressing rich behavioral semantics [10].

The fUML is a subset of the standard UML for which a standard execution constraint language, PSL, is used to define the semantics of the behavior model [11]. UML 2.0 has a common behavioral concept that supports the foundation for its three diagrams: activities, state machines, and interactions. Although fUML provides behavioral constraints to make the model executable, it cannot provide the behavioral semantics of invoking an activity or calling a state machine or invoking an interaction simultaneously. In other words, fUML can only specify one or two, but all three behavioral diagrams at a time.

The Alf is a complementary specification to fUML [12]. The key use of Alf is to act as the notation for specifying executable behaviors in UML, for example, methods for object operations, the behavior of an object, or transition effects on state machines. Like fUML, Alf also fails to provide the behavioral semantic constraint of invoking an activity or calling a state machine or invoking an interaction at a time.

In order to overcome the shortcomings of the current UML 2.0 metamodel solutions, we need to develop an integrated semantics framework that is able to unify structural constructs with behavioral semantic constraints of invoking an activity or calling a state machine or invoking an interaction. SBC-ASM MBL is such a candidate. In SBC-ASM MBL, the structural constructs and behavioral semantic constraints of invoking an activity or calling a state machine or invoking an interaction are unified.

As shown in Fig. 2, the MDE process using the SBC metamodel solution consists of two subprocesses: 1) requirements analysis and 2) transformation from SBC-ASM to UML user model. The input and output of the "requirements analysis" subprocess are "stakeholder interviews" and "SBC-ASM," respectively. The "transformation from SBC-ASM to UML user model" subprocess takes SBC-ASM as input and generates UML user model diagrams as output.

In model-driven software engineering, software verification and validation (V&V) is the process of checking whether a software system fulfills its requirements and whether it meets its intended purpose [14], [15]. Software V&V is also called software quality control. If it takes a lot of effort to check the user model consistency, it will be difficult to implement software verification and validation. The SBC metamodel solution greatly reduces the work of checking user model inconsistencies. Therefore, using the SBC metamodel solution is preferable to the current UML 2.0 metamodel solution regarding software validation and verification.

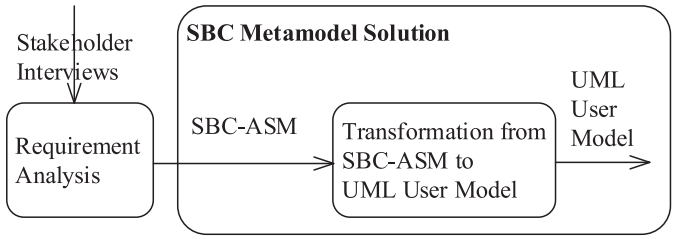

Fig. 2. MDE process using the SBC metamodel solution.

III. SBC-ASM MBL

A. Operation-Based Value-Passing Interactions

The object is the fundamental modular unit for describing software structure in UML 2.0 [1]–[3]. An operation represents a procedure, method, or function that an object performs when a caller calls it. Each operation defines a set of parameters that describes the arguments passed in with the request, or passed back out once a request has been handled. An operation (can be extended to operation call or operation return) signature is a combination of its name along with parameters as follows:

<operation name> (<parameter list>)

The parameters in the parameter list represent the inputs or outputs of the operation. Each parameter in the list is displayed with the following format:

<direction> <parameter name> : <parameter type>

Parameter direction may be in, out, or in–out. We formally describe the "operation call or operation return signature" as a relation $L \subseteq O \times P$ where O is a set of "operation names" and P is a set of "parameter lists."

An interaction [9] represents the indivisible and instantaneous message exchange between the caller agent (either external environment's actor or object) and the callee agent (object). In the operation-based value-passing approach as shown in Fig. 3, the caller agent sends (or receives) the message to (or from) the callee object through the operation call or operation return interaction. The solid line indicates an operation call from the caller agent to the callee object, and the dashed line indicates an operation return from the callee object to the caller agent. In the figure, getPastDueBalance(in studentId: String) is an operation call signature and getPastDueBalance(out PastDueBalance: Real) is an operation return signature. The operation call signature and its corresponding operation return signature can be

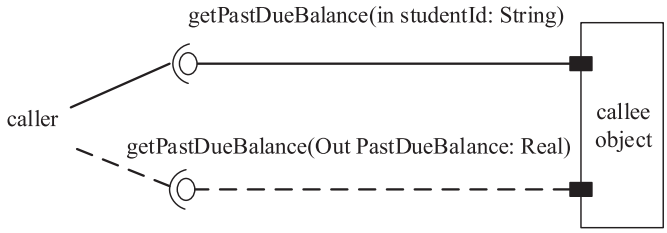

Fig. 3. Operation-based value-passing interactions.

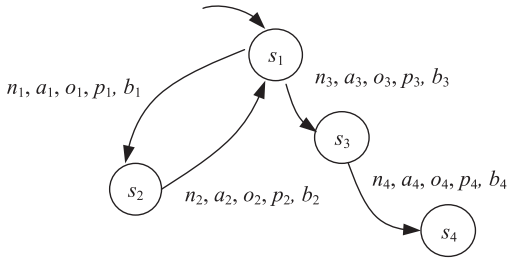

Fig. 4. Diagram of the SBC-ASM ASM_{01}.

merged into an operation signature (e.g., getPastDueBalance(in studentId: String; out PastDueBalance: Real). Fig. 3 also depicts that the "getPastDueBalance" operation is *required* by the caller and is *provided* by the callee object.

We formally describe the "operation-based value-passing interaction" as a relation $\Delta \subseteq N \times A \times O \times P \times B$, where N is a set of "operation call or operation return tags" and A is a set of "external environment's actors or objects" and O is a set of "operation names" and P is a set of "parameter lists" and B is a set of "objects."

B. SBC Abstract State Machine

In SBC-ASM MBL, we use the SBC-ASM as a single diagram to specify the semantics of a system. The SBC-ASM is a labeled transition system [16]. Overall, the SBCASM provides the integrated semantics framework for the integration of structural constructs with behavioral constructs. The notion of a SBCASM is defined as follows.

Definition 1: (ASM) A SBC-ASM $ASM = (S, N, A, O, P, B, ASMR)$ consists of

1) a finite set S of states,
2) a finite set N of operation call or operation return tags,
3) a finite set A of external environment's actors or objects,
4) a finite set O of operation names,
5) a finite set P of parameter lists,
6) a finite set B of objects,
7) a transition relation ASMR$\subseteq S_1 \times N \times A \times O \times P \times B \times S_2$, where $(s_j, n, a, o, p, b, s_k) \in$ ASMR is written as $s_j \xrightarrow{n,a,o,p,b} s_k$.

We draw a diagram to represent the SBC-ASM. Fig. 4 shows the diagram of the SBC-ASM ASM_{01}. In this diagram SBC-ASM, the state is represented by a circle; the edge is used to represent the labeled "transition" between the two states; the initial state is usually represented by an arrow with no origin pointing to the state. In SBC-ASM, each transition is labeled with an "operation-based value-passing interaction" message exchange, represented as "(n, a, o, p, b)," that causes the transition. In a state, if there are multiple transitions, it can nondeterministically choose any one to continue.

TABLE I
RELATION $ASMR_{01}$ OF THE ASM ASM_{01}

S_1	N	A	O	P	B	S_2
s_1	n_1	a_1	o_1	p_1	b_1	s_2
s_2	n_2	a_2	o_2	p_2	b_2	s_1
s_1	n_3	a_3	o_3	p_3	b_3	s_3
s_3	n_4	a_4	o_4	p_4	b_4	s_4

We can also list the relationships that represent the SBC-ASM. Table I shows the transition relation $ASMR_{01}$ of the SBC-ASM ASM_{01}.

In order to reduce the complexity of the ASM, we shall introduce an orthogonal composite state. An orthogonal composite state in the ASM may have many regions, which may each contain substates. These regions are orthogonal to each other. When an orthogonal composite state is active, each region has its own active state that is independent of the others and any incoming interaction is independently analyzed within each region. We use $ASM_1 \| ASM_2 \| ASM_3 \| \ldots \| ASM_m$ to represent an orthogonal composite state, which means the composition of $ASM_1, ASM_2, ASM_3, \ldots$, and ASM_m.

In SBC-ASM MBL, the SBC ASM of a system ASM_{system}, standing for the system as a whole, is defined as $\|_{i=1,m} ASM_i$ or $ASM_1 \| ASM_2 \| \ldots \| ASM_m$. Each SBC ASM ASM_i, standing for the ith behavior of the system, represented by a transition relation $ASMR_i \subseteq S_1 \times A \times O \times P \times B \times S_2$, where $(s_{ij}, n, a, o, p, b, s_{ik}) \in ASMR_i$ is denoted by $s_{ij} \xrightarrow{n,a,o,p,b} s_{ik}$. The SBC-ASM of a system ASM_{system} is represented by the transition relation $ASMR_{\text{system}}$ which is defined as $\|_{i=1,\text{m}} ASMR_i$ or $ASMR_1 \| ASMR_2 \| \ldots \| ASMR_m$.

We can draw a diagram to represent the SBC-ASM of a software system. Fig. 5 shows the diagram of the SBC-ASM ASM_{system}.

We can also list the relationships that represent the ASM of a software system. Table II shows the transition relation $ASMR_{\text{system}}$ of the ASM ASM_{system}.

IV. SBC Metamodel Soluton

A. Model-Driven Engineering Process

The complete UML user model process starts from the design of functional models, which includes the use case descriptions and activity diagrams for each use case, follows by the design of structure models (class diagrams) and behavior models (sequence diagrams), and finally the ER models (data models).

The MDE process using the SBC metamodel solution consists of two subprocesses: 1) requirements analysis and 2) transformation from SBC-ASM to UML user model.

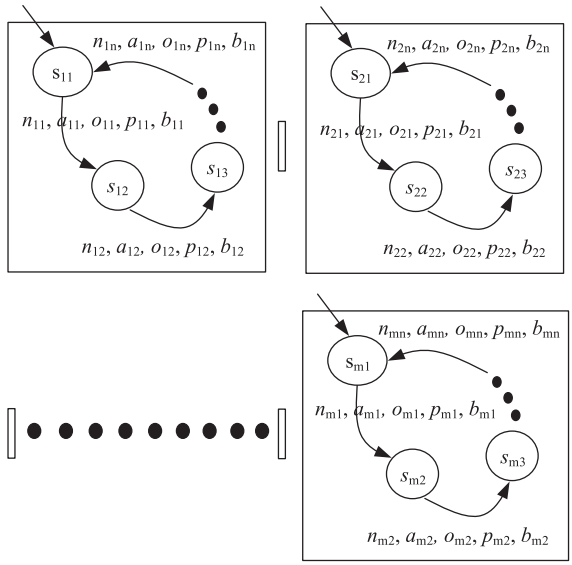

Fig. 5. SBC-ASM ASM_{system}.

TABLE II
Relation $\text{ASMR}_{\text{SYSTEM}}$

S_1	N	A	O	P	B	S_2
s_{11}	n_{11}	a_{11}	o_{11}	p_{11}	b_{11}	s_{12}
s_{12}	n_{12}	a_{12}	o_{12}	p_{12}	b_{12}	s_{13}
•	•	•	•	•	•	•
s_{1n}	n_{1n}	a_{1n}	o_{1n}	p_{1n}	b_{1n}	s_{11}

S_1	N	A	O	P	B	S_2
s_{21}	n_{21}	a_{21}	o_{21}	p_{21}	b_{21}	s_{22}
s_{22}	n_{22}	a_{22}	o_{22}	p_{22}	b_{22}	s_{23}
•	•	•	•	•	•	•
s_{2n}	n_{2n}	a_{2n}	o_{2n}	p_{2n}	b_{2n}	s_{21}

S_1	N	A	O	P	B	S_2
s_{m1}	n_{m1}	a_{m1}	o_{m1}	p_{m1}	b_{m1}	s_{m2}
s_{m2}	n_{m2}	a_{m2}	o_{m2}	p_{m2}	b_{m2}	s_{m3}
•	•	•	•	•	•	•
s_{mn}	n_{mn}	a_{mn}	o_{mn}	p_{mn}	b_{mn}	s_{m1}

The output of the "requirements analysis" subprocess is SBC-ASM. The "transformation from SBC-ASM to UML user model" subprocess takes SBC-ASM as input and generates UML user model, such as object diagrams, class diagrams, composite structure diagrams, component diagrams, use case diagrams, activity diagrams, state diagrams, sequence diagrams, communication diagram, etc., as output.

It is assumed that all UML user model diagrams can be projected from SBC-ASM. However, in this article, we only demonstrate three projections. The SBC-ASM transition relation $ASMR_{\text{system}}$ created through the "requirements analysis" process will first be stored into the database, and then transformed to the class diagram, state diagram, and sequence diagram, respectively.

B. Projecting the Class Diagram From the SBC-ASM

In UML 2.0, the class diagram is a static structure diagram that describes the structure of a software system by showing the classes of the software system, their attributes, operations (or methods), and relationships between objects. The notion of a UML 2.0 class diagram is defined as follows.

Definition 2: (CLASS DIAGRAM) A class diagram $ClsD = (C, O, P, ClsDR)$ consists of

1) a finite set C of classes,
2) a finite set O of operation names,
3) a finite set P of parameter lists,
4) a relation $\text{ClsDR} \subseteq C \times O \times P$, where $(c, o, p) \in \text{ClsDR}$.

The UML 2.0 class diagram of a software system $ClsD_{\text{system}}$ is represented by a relation $ClsDR_{\text{system}} \subseteq C \times O \times P$, where $(c, o, p) \in ClsDR_{\text{system}}$, shown in Table III.

TABLE III
Relation $\text{CLSDR}_{\text{SYSTEM}}$

C	O	P
c_1	o_1	p_1
c_1	o_2	p_2
c_4	o_4	p_4
•	•	•
c_n	o_n	p_n

ALGORITHM 1: (Projecting $ClsDR_{\text{system}}$ From $ASMR_{\text{system}}$).

For $i = 1$, m **Loop**
SELECT DISTINCT B, O, P INTO $ClsDR_i$ (C, O, P) FROM $ASMR_i$ WHERE $N =$ 'OPERATION_CALL';
SELECT DISTINCT B, O, P INTO $ClsDR\_RETURN_i$ (C, O, P) FROM $ASMR_i$ WHERE $N =$ 'OPERATION_RETURN';
UPDATE $ClsDR_i$ SET O, $P =$ MERGE (Operation Call Signature, Operation Return Signature) WHERE there exists corresponding Operation Return Signature in $ClsDR\_RETURN_i$;
INSERT INTO $ClsDR_{\text{i}\sim\text{m}}$ (C, O, P) SELECT $*$ FROM $ClsDR_{\text{i}}$;
End Loop;
SELECT DISTINCT $*$ INTO $ClsDR_{\text{system}}$ FROM $ClsDR_{\text{i}\sim\text{m}}$
END ALGORITHM

The algorithm used to project the ClsD relation $ClsDR_{\text{system}} \subseteq C \times O \times P$ from the ASM relation $ASMR_{\text{system}} \subseteq S_1 \times N \times A \times O \times P \times B \times S_2$ is as follows. In the algorithm, for the loop part, it selects into $ClsDR_i$ from $ASMR_i$ with operation call or return tag. After that, it will update $ClsDR_i$ by merging the information from $ASMR_i$ and then insert into $ClsDR_{\text{i}\sim\text{m}}$ from $ClsDR_i$.

Once we have the ClsD relation $ClsDR_{\text{system}}$, it is easy to get a UML 2.0 class diagram of the software system.

C. Projecting the State Diagram From the SBC-ASM

In UML 2.0, the state diagram represents behavior of a software system in terms of its transition between states triggered by actions. The notion of a UML 2.0 state diagram is defined as follows.

TABLE IV
Relation $StDR_{\text{SYSTEM}}$

S_1	N	O	S_2
s_{11}	n_{11}	o_{11}	s_{12}
s_{12}	n_{12}	o_{12}	s_{13}
•	•	•	•
s_{1n}	n_{1n}	o_{1n}	s_{11}

‖

S_1	N	O	S_2
s_{21}	n_{21}	o_{21}	s_{22}
s_{22}	n_{22}	o_{22}	s_{23}
•	•	•	•
s_{2n}	n_{2n}	o_{2n}	s_{21}

‖ • • • • • ‖

S_1	N	O	S_2
s_{m1}	n_{m1}	o_{m1}	s_{m2}
s_{m2}	n_{m2}	o_{m2}	s_{m3}
•	•	•	•
s_{mn}	n_{mn}	o_{mn}	s_{m1}

TABLE V
RELATION $StDR_{\text{SYSTEM}}$

E	N	A	O	P	B
e_{11}	n_{11}	a_{11}	o_{11}	p_{11}	b_{11}
e_{12}	n_{12}	a_{12}	o_{12}	p_{12}	b_{12}
•	•	•	•	•	•
e_{1n}	n_{1n}	a_{1n}	o_{1n}	p_{1n}	b_{1n}

‖

E	N	A	O	P	B
e_{21}	n_{21}	a_{21}	o_{21}	p_{21}	b_{21}
e_{22}	n_{22}	a_{22}	o_{22}	p_{22}	b_{22}
•	•	•	•	•	•
e_{2n}	n_{2n}	a_{2n}	o_{2n}	p_{2n}	b_{2n}

‖ • • • • • • • • ‖

E	N	A	O	P	B
e_{m1}	n_{m1}	a_{m1}	o_{m1}	p_{m1}	b_{m1}
e_{m2}	n_{m2}	a_{m2}	o_{m2}	p_{m2}	b_{m2}
•	•	•	•	•	•
e_{mn}	n_{mn}	a_{mn}	o_{mn}	p_{mn}	b_{mn}

Definition 3: (STATE DIAGRAM) A state diagram $StD = (S, N, O, StDR)$ consists of

1) a finite set S of states,
2) a finite set N of operation call or operation return tags,
3) a finite set O of operation names,
4) a relation StDR$\subseteq S \times N \times O \times S$, where $(s_j, n, o, s_k) \in$StDR is denoted by $s_j \xrightarrow{n,o} s_k$.

The UML 2.0 state diagram of a software system StD_{system} is defined as $\|_{i=1,m}\ StD_i$ or $StD_1 \| StD_2 \| \ldots \| StD_m$. Each state diagram StD_i is represented by a relation $StDR_i \subseteq S_1 \times N \times O \times S_2$, where $(s_{ij}, n, o, s_{ik}) \in StDR_i$ is denoted by $s_{ij} \xrightarrow{n,o} s_{ik}$. The state diagram of a software system StD_{system} is represented by the relation $StDR_{\text{system}}$, which is defined as $\|_{i=1,m}\ StDR_i$ or $StDR_1 \| StDR_2 \| \ldots \| StDR_m$, as shown in Table IV.

The algorithm used to project the StD relation $StDR_{\text{system}} \subseteq S_1 \times N \times O \times S_2$ from the ASM relation $ASMR_{\text{system}} \subseteq S_1 \times N \times A \times O \times P \times B \times S_2$ is as follows. In the algorithm, for the loop part, it selects into $StDR_i$ from $ASMR_i$.

Once we have the StD relation $StDR_{\text{system}}$, it is easy to get a UML 2.0 state diagram of the software system.

ALGORITHM 2: (Projecting $StDR_{\text{system}}$ From $ASMR_{\text{system}}$).

For i = 1, m **Loop**
SELECT S_1, N, O, S_2 INTO $StDR_i$ FROM $ASMR_i$;
End Loop;
ORTHOGONALLY COMPOSE ALL $StDR_i$ (i.e., $\|_{i=1,m}\ StDR_i$) TO GET $StDR_{\text{system}}$
END ALGORITHM

ALGORITHM 3: (Projecting $SqDR_{\text{system}}$ From $ASMR_{\text{system}}$).

For i = 1, m **Loop**
CREATE RELATION $SqDR_i$ (E int IDENTITY(1,1), N, A, O, P, B);
INSERT INTO $SqDR_i$ (N, A, O, P, B) SELECT N, A, O, P, B FROM $ASMR_i$;
End Loop;
ORTHOGONALLY COMPOSE ALL $SqDR_i$ (i.e., $\|_{i=1,m}\ SqDR_i$) TO GET $SqDR_{\text{system}}$
END ALGORITHM

D. Projecting the Sequence Diagram From the SBC-ASM

In UML 2.0, the state diagram represents behavior of a software system in terms of its transition between states triggered by actions. The notion of a UML 2.0 state diagram is defined as follows.

Definition 4: (SEQUENCE DIAGRAM) A sequence diagram $SqD = (E, N, A, O, P, B, SqDR)$ consists of

1) a finite set E of execution orders,
2) a finite set N of operation call or operation return tags,
3) a finite set A of external environment's actors or objects,
4) a finite set O of operation names,
5) a finite set P of parameter lists,
6) a finite set B of objects,
7) a relation SqDR$\subseteq E \times N \times A \times O \times P \times B$, where $(e, n, a, o, p, b) \in$ SqDR.

The UML 2.0 sequence diagram of a software system SqD_{system} is defined as $\|_{i=1,m}\ SqD_i$ or $SqD_1 \| SqD_2 \| \ldots \| SqD_m$. Each sequence diagram SqD_i is represented by a relation $SqDR_i \subseteq E \times N \times A \times O \times P \times B$, where $(e, n, a, o, p, b) \in SqDR_i$. The sequence diagram of a software system SqD_{system} is represented by the relation $SqDR_{\text{system}}$, which is defined as $\|_{i=1,m}\ SqDR_i$ or $SqDR_1 \| SqDR_2 \| \ldots \| SqDR_m$, as shown in Table V.

The algorithm used to project the SqD relation $SqDR_{\text{system}} \subseteq E \times N \times A \times O \times P \times B$ from the ASM relation $ASMR_{\text{system}} \subseteq S_1 \times N \times A \times O \times P \times B \times S_2$ is as follows. In the algorithm, for the loop part, a new $SqDR_i$ will be created first. After that, it will select information from $ASMR_i$ and then insert the selected information into $SqDR_i$.

Once we have the SqD relation $SqDR_{\text{system}}$, it is easy to get a UML 2.0 sequence diagram of the software system.

V. CASE: ONLINE SHOPPING SYSTEM (OSS)

A. Online Shopping System

The scenario of the OSS contains three behaviors. In the behavior of *Make_Order_Request*, a customer can request to order one or more items from the supplier. The customer provides personal details such as address and credit card information. This information is stored in the customer's account. If the credit card is valid, a delivery order is created and sent to the supplier. In the behavior of *Confirm_Shipment_and_Bill_Customer*, the supplier prepares the shipment manually and confirms that the order is ready for shipment. When the order is shipped, the customer is notified and the customer's credit card account is charged. In the behavior of *View_Order*, the customer will view the details of the delivery order.

B. ASM of the OSS

Once the requirements of the OSS are established, the modeler can use these requirements to create an SBC-ASM for the OSS. The SBC-ASM of the OSS is described by a SBC abstract state machine ASM_{OSS} (defined as "$ASM_1 \| ASM_2 \| ASM_3$") with the transition relation $ASMR_{\mathrm{OSS}} \subseteq S_1 \times N \times A \times O \times P \times B \times S_2$ (defined as "$ASMR_1 \| ASMR_2 \| ASMR_3$") as shown in Table VI. In the SBC-ASM ASM_{OSS}, ASM_1 represents the *Make_Order_Request* behavior, ASM_2 represents the *Confirm_Shipment_and_Bill_Customer* behavior, and ASM_3 represents the *View_Order* behavior.

C. Projecting the Class Diagram of the OSS

We apply the algorithm of projecting the ClsD relation (i.e., $ClsDR_{\mathrm{OSS}}$) from the ASM relation (i.e., $ASMR_{\mathrm{OSS}}$) of the OSS. After the projection, we get the relation $ClsDR_{\mathrm{OSS}} \subseteq C \times O \times P$ as shown in Table VII.

From the projected ClsD relation $ClsDR_{\mathrm{OSS}}$, we draw the corresponding UML 2.0 class diagram of the OSS, as shown in Fig. 6.

D. Projecting the State Diagram of the OSS

We apply the algorithm of projecting the StD relation (i.e., $StDR_{\mathrm{OSS}}$) from the ASM relation (i.e., $ASMR_{\mathrm{OSS}}$) of the OSS. After the projection, we get the relation $StDR_{\mathrm{OSS}} \subseteq S_1 \times N \times O \times S_2$ as shown in Table VIII.

From the projected StD relation $StDR_{\mathrm{OSS}}$, we draw the corresponding UML 2.0 state diagram of the OSS, as shown in Fig. 7.

E. Projecting the Sequence Diagram of the OSS

We apply the algorithm of projecting the SqD relation (i.e., $SqDR_{\mathrm{OSS}}$) from the ASM relation (i.e., $ASMR_{\mathrm{OSS}}$) of the OSS. After the projection, we get the relation $SqDR_{\mathrm{OSS}} \subseteq E \times N \times A \times O \times P \times B$ as shown in Table IX.

From the projected SqD relation $SqDR_{\mathrm{OSS}}$, we draw the corresponding UML 2.0 sequence diagram of the OSS, as shown in Fig. 8.

TABLE VI
RELATION $ASMR_{\mathrm{OSS}}$

S_1	N	A	O	P	B	S_2
s_{11}	*CAL*	Customer	Request_ Order_ from_ Customer	in Request_ Order_ Info	:Customer_ UI	s_{12}
s_{12}	*CAL*	:Customer_ UI	Request_ Order_ from_UI	in Request_ Order_ Info	:Customer_ Coordinator	s_{13}
s_{13}	*CAL*	:Customer_ Coordinator	Authorize_ Credit_ Card_ Charge	in Credit_ Card_Id; in Amount; out Authorization_ Response	:Credit_ Card_ Service	s_{14}
s_{14}	*CAL*	:Customer_ Coordinator	Store_ Order	in Order; out Order_Id	:Delivery_ Order_ Service	s_{15}
s_{15}	*RET*	:Customer_ UI	Request_ Order_ from_ UI	out Order_Info	:Customer_ Coordinator	s_{16}
s_{16}	*RET*	Customer	Request_ Order_ from_ Customer	out Order_Info	:Customer_ UI	s_{11}
‖						
S_1	N	A	O	P	B	S_2
s_{21}	*CAL*	Supplier	Shipping	in Order_Id	:Supplier_ UI	s_{22}
s_{22}	*CAL*	:Supplier_ UI	Ready_ for_ Shippment	in Order_Id	:Supplier_ Coordinator	s_{23}
s_{23}	*CAL*	:Supplier_ Coordinator	Request_ Invoice	in Order_Id; out Invoice	:Delivery_ Order_ Service	s_{24}
s_{24}	*CAL*	:Supplier_ Coordinator	Commit_ Credit_ Card_ Charge	in Credit_ Card_Id; in Amount; out Commit_ Response	:Credit_ Card_ Service	s_{25}
s_{25}	*CAL*	:Supplier_ Coordinator	Confirm_ Payment	in Credit_ Order_Id; in Amount; out Order_ Status	:Delivery_ Order_ Service	s_{26}
s_{26}	*RET*	:Supplier_ UI	Ready_ for_ Shippment	out Order_ Status	:Supplier_ Coordinator	s_{27}
s_{27}	*RET*	Supplier	Shipping	out Order_ Status	:Supplier_ UI	s_{21}
‖						
S_1	N	A	O	P	B	S_2
s_{31}	*CAL*	Customer	Request_ Order_ Status_ from_ Customer	in Order_Id	:Customer_ UI	s_{32}
s_{32}	*CAL*	:Customer_ UI	Request_ Order_ Status_ from_UI	in Order_Id	:Customer_ Coordinator	s_{33}
s_{33}	*CAL*	:Customer_ Coordinator	Read_Order	in Order_Id; out Order	:Delivery_ Order_ Service	s_{34}
s_{34}	*RET*	:Customer_ UI	Request_ Order_ Status_ from_UI	out Order_ Status	:Customer_ Coordinator	s_{35}
s_{35}	*RET*	Customer	Request_ Order_ Status_ from_ Customer	out Order_ Status	:Customer_ UI	s_{31}

TABLE VII
Relation $CLsDR_{OSS}$

C	O	P
:Customer_ UI	Request_Order_ from_Customer	in Request_Order_Info; out Order_Info
:Customer_ UI	Request_Order_ Status_from_ Customer	in Order_Id; out Order_Status
:Supplier_ UI	Shipping	in Order_Id; out Order_Status
:Customer_ Coordinator	Request_ Order_ from_UI	in Request_Order_Info; out Order_Info
:Customer_ Coordinator	Request_Order_ Status_from_UI	in Order_Id; out Order_Status
:Supplier_ Coordinator	Ready_for_ Shippment	in Order_Id

C	O	P
:Credit_ Card_ Service	Authorize_ Credit_Card_ Charge	in Credit_Card_Id; in Amount; out Authorization_ Response
:Credit_ Card_ Service	Commit_ Credit_Card_ Charge	in Credit_Card_Id; in Amount; out Commit_Response
:Delivery_ Order_ Service	Store_ Order	in Order; out Order_Id
:Delivery_ Order_ Service	Request_ Invoice	in Order_Id; out Invoice
:Delivery_ Order_ Service	Confirm_ Payment	in Credit_Order_Id; in Amount; out Order_Status
:Delivery_ Order_ Service	Read_Order	in Order_Id; out Order

TABLE VIII
RELATION $StDR_{OSS}$

S_1	N	O	S_2
s_{11}	*CAL*	Request_Order_from_Customer	s_{12}
s_{12}	*CAL*	Request_Order_from_UI	s_{13}
s_{13}	*CAL*	Authorize_Credit_Card_Charge	s_{14}
s_{14}	*CAL*	Store_Order	s_{15}
s_{15}	*RET*	Request_Order_from_UI	s_{16}
s_{16}	*RET*	Request_Order_from_Customer	s_{11}

S_1	N	O	S_2
s_{21}	*CAL*	Shipping	s_{22}
s_{22}	*CAL*	Ready_for_Shippment	s_{23}
s_{23}	*CAL*	Request_Invoice	s_{24}
s_{24}	*CAL*	Commit_Credit_Card_Charge	s_{25}
s_{25}	*CAL*	Confirm_Payment	s_{26}
s_{26}	*RET*	Ready_for_Shippment	s_{27}
s_{27}	*RET*	Shipping	s_{21}

S_1	N	O	S_2
s_{31}	*CAL*	Shipping	s_{32}
s_{32}	*CAL*	Ready_for_Shippment	s_{33}
s_{33}	*CAL*	Request_Invoice	s_{34}
s_{34}	*RET*	Ready_for_Shippment	s_{35}
s_{35}	*RET*	Shipping	s_{31}

VI. Effort Evaluation of SBC Metamodel Soluton

A. Evaluation of Efforts to Construct SBC-ASM

The output of the "requirements analysis" subprocess of the MDE process using the SBC metamodel solution is the construction of SBC-ASM. That is, after the "requirements analysis," the requirements will be documented by the SBC-ASM MBL.

TABLE IX
Relation $SQDR_{\mathrm{OSS}}$

E	N	A	O	P	B
1	*CAL*	Customer	Request_ Order_ from_ Customer	in Request_ Order_ Info	:Customer_ UI
2	*CAL*	:Customer_ UI	Request_ Order_ from_UI	in Request_ Order_ Info	:Customer_ Coordinator
3	*CAL*	:Customer_ Coordinator	Authorize_ Credit_ Card_ Charge	in Credit_ Card_Id; in Amount; out Authorization_ Response	:Credit_ Card_ Service
4	*CAL*	:Customer_ Coordinator	Store_ Order	in Order; out Order_Id	:Delivery_ Order_ Service
5	*RET*	:Customer_ UI	Request_ Order_ from_UI	out Order_Info	:Customer_ Coordinator
6	*RET*	Customer	Request_ Order_ from_ Customer	out Order_Info	:Customer_ UI

$\square$

E	N	A	O	P	B
1	*CAL*	Supplier	Shipping	in Order_Id	:Supplier_ UI
2	*CAL*	:Supplier_ UI	Ready_ for_ Shippment	in Order_Id	:Supplier_ Coordinator
3	*CAL*	:Supplier_ Coordinator	Request_ Invoice	in Order_Id; out Invoice	:Delivery_ Order_ Service
4	*CAL*	:Supplier_ Coordinator	Commit_ Credit_ Card_ Charge	in Credit_ Card_Id; in Amount; out Commit_ Response	:Credit_ Card_ Service
5	*CAL*	:Supplier_ Coordinator	Confirm_ Payment	in Credit_ Order_Id; in Amount; out Order_ Status	:Delivery_ Order_ Service
6	*RET*	:Supplier_ UI	Ready_ for_ Shippment	out Order_ Status	:Supplier_ Coordinator
7	*RET*	Supplier	Shipping	out Order_ Status	:Supplier_ UI

$\square$

E	N	A	O	P	B
1	*CAL*	Customer	Request_ Order_ Status_ from_ Customer	in Order_Id	:Customer_ UI
2	*CAL*	:Customer_ UI	Request_ Order_ Status_ from_UI	in Order_Id	:Customer_ Coordinator
3	*CAL*	:Customer_ Coordinator	Read_Order	in Order_Id; out Order	:Delivery_ Order_ Service
4	*RET*	:Customer_ UI	Request_ Order_ Status_ from_UI	out Order_ Status	:Customer_ Coordinator
5	*RET*	Customer	Request_ Order_ Status_ from_ Customer	out Order_ Status	:Customer_ UI

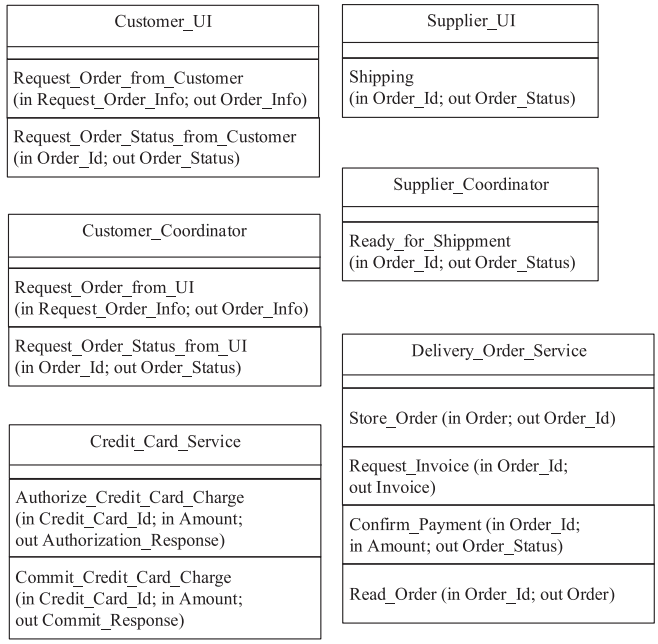

Fig. 6. Projected ClsD view of the OSS.

The construction of SBC-ASM goes through the following steps.

Step 1: Interviewing Stakeholders

Step 2: Identifying behaviors (e.g., *Make_Order_Request*, *Confirm_Shipment_and_Bill_Customer*, *View_Order* in the OSS case study).

Step 3: Identifying actors (e.g., Customer, Supplier in the OSS case study).

Step 4: Identifying objects (e.g., :Customer_UI in the OSS case study).

Step 5: Identifying interactions between actors and objects (e.g., <*CAL*, Customer, Request_Order_from_Customer, in Request_Order_Info, :Customer_UI> in the OSS case study).

Step 6: Identifying interactions between internal objects (e.g., <*CAL*, :Customer_UI, Request_Order_from_UI, in Request_Order_Info, :Customer_Coordinator> in the OSS case study).

Step 7: Documenting each $ASMR_i$.

Step 8: Documenting $ASM_1 \square SM_2 \square \ldots \square ASM_{\mathrm{m}}$.

The "behavior" in SBC-ASM is similar to the "use case" in the UML user model. The mechanism for identifying use cases can be applied to step 2 of identifying behaviors in the construction of SBC-ASM. Identifying actors in step 3 is to recognize which external entities the system must interact with. Objects are interrelated entities that make up a system. Step 4 is to identify these objects in the system. An interaction represents the message exchanges between caller agent (actor or object) and callee objects. There are two types of interactions. The interaction between the actor and the object will be identified in step 5. Step 6 will identify the interaction between objects. After all interactions are identified, each $ASMR_i$ will be documented

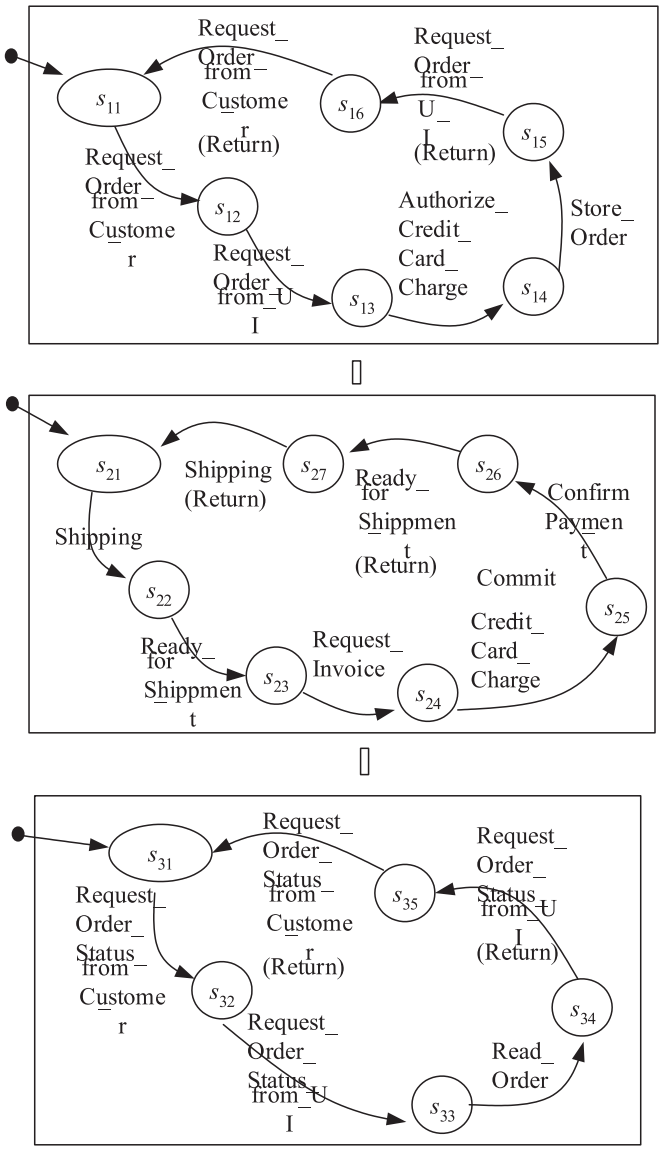

Fig. 7. Projected StD view of the OSS.

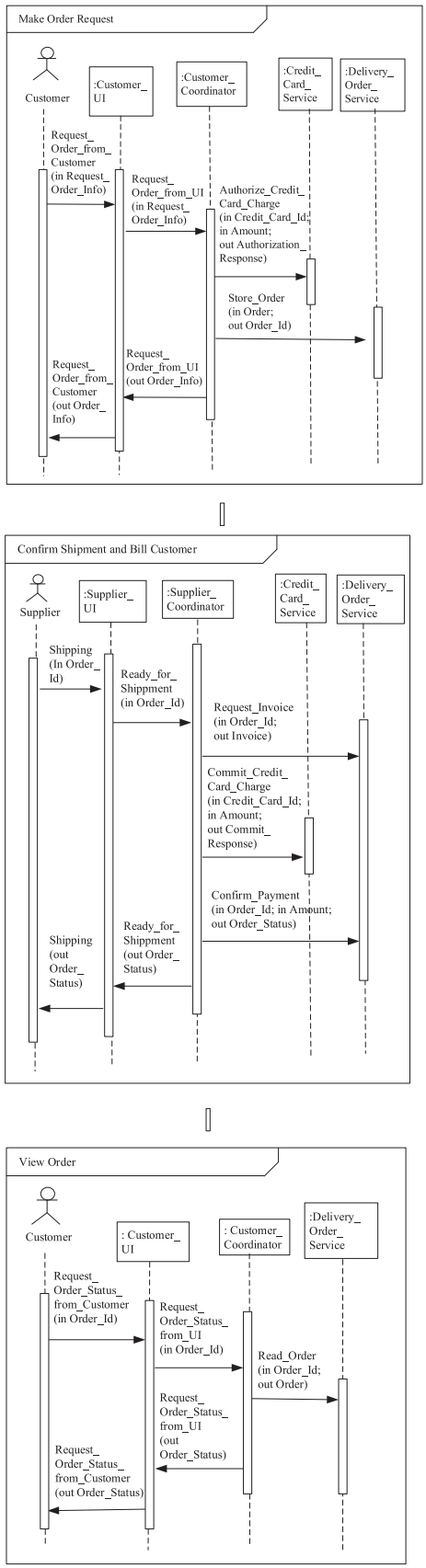

Fig. 8. Projected SqD view of the OSS.

in step 7. After documenting each $ASMR_i$, step 8 will apply the orthogonal composition to obtain $ASM_1 [] ASM_2 [] \ldots [] ASM_{\mathrm{m}}$.

The way SBC-ASM captures the static and dynamic aspects is very simple, just like the object-oriented analysis and design of the ICONIX process [17], [18]. The ICONIX approach achieves its robustness analysis by capturing the message exchange between objects. In summary, SBC-ASM uses steps 3 and 4 to capture the static aspect of the system. As a dynamic aspect of the system, SBC-ASM uses steps 2, 5, and 6 to capture the exchange of messages between caller agents (actor or objects) and callee objects.

B. Evaluation of Efforts to Transform SBC-ASM Into UML User Model

Since SBC-ASM forms an integrated UML user model, it is assumed that all UML user model diagrams can be transformed

from SBC-ASM. We treat SBC-ASM as a "base" relation; we also treat each UML user model diagram as a "derived" relation. The general concept of transforming each UML user model diagram from SBC-ASM is to apply relational operations to the "base" SBC-ASM relation and obtain the "derived" UML user model diagram relation.

In this article, we only demonstrate three transformations. The SBC-ASM "base" relation $ASMR_{\text{system}}$ created through the "requirements analysis" process will first be stored into the database, and then transformed to the "derived" class diagram relation, state diagram relation and sequence diagram relation, respectively.

The three algorithms shown in Section VI are based on relational operations and can be easily implemented as programs. Once the "base" relation $ASMR_{\text{system}}$ is created, the transformation of the "derived" class diagram relation, state diagram relation, and sequence diagram relation can be automatically performed without any effort.

VII. Conclusion

A. Contributions of This Article

In this article, SBC-ASM MBL is proposed as a metamodel solution for UML 2.0 in MDE. One important use of the UML 2.0 metamodel is to ensure model consistency between different diagrams in the user model. Nowadays, most current UML 2.0 metamodeling approaches are based on the MOF, which includes OCL, fUML, the Alf, etc. OCL specifies behavioral semantics by pre- and post- conditions on operations that do not allow altering the state of a model. Therefore, OCL is insufficient to express rich behavioral semantics. fUML cannot provide the behavioral semantics of invoking an activity or calling a state machine or invoking an interaction at the same time. In other words, fUML can only specify one or two, but all three behavioral diagrams at a time. The Alf also fails to provide the behavioral semantic constraint of invoking an activity or calling a state machine or invoking an interaction at a time.

In order to overcome the shortcomings of current UML 2.0 metamodel approaches, we need an integrated semantic framework that is able to unify the structural and behavioral constructs. Adopting SBC-ASM MBL as a metamodel solution for UML 2.0, we use the SBC-ASM as a single diagram to complete the overall semantic specification of the software system. Through the SBC-ASM and its corresponding transition relation, each diagram in the UML 2.0 user model can be projected as a view of the SBC-ASM. Therefore, we conclude that the ASM used by the SBC-ASM MBL as a metamodel solution for UML 2.0 indeed provides an integrated semantic framework to ensure UML 2.0 model consistency.

B. Limitation of the SBC-ASM MBL and Future Work

So far, the SBC-ASM MBL proposed in this article does not articulate the applicability of the SBC-ASM in the context of System-of-Systems Engineering (SoSE). This is the limitation of the current SBC-ASM MBL approach.

As a general application that supports system requirements, design, analysis, verification, and validation tasks beginning in the conceptual design phase and continuing throughout development and later life cycle phases, SoSE aims to promote systems engineering activities that have traditionally been performed using the document-based approach and result in enhanced specification and design quality, reuse of system specification and design artifacts, as well as communication between development teams [19], [20].

The core theme of SoSE is a kernel model, i.e., SysML, of the system's (static) structure and (dynamic) behavior, with an emphasis on using model-based methods and tools to develop and improve the model.

Since SBC-ASM MBL provides a good metamodel solution for UML 2.0, it can also provide a good metamodel solution for SysML, that is, using only a single diagram to specify the semantics of the system. Therefore, when the SBC-ASM MBL is used for the SysML metamodel, the consistency of the model will be fully guaranteed in SoSE modeling. This will be our future work on the SBC-ASM MBL.

Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable comments, which help clarify subtle points and triggered new ideas.

References

[1] D. C. Schmidt, "Model-driven engineering," *IEEE Comput.*, vol. 39, no. 2, pp. 25–31, 2006.
[2] M. R. Blaha and J. R. Rumbaugh, *Object-Oriented Modeling and Design with UML*, 2nd Ed., Upper Saddle River, NJ: Pearson, 2004.
[3] M. Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, Boston: Addison-Wesley, 2016.
[4] T. Weilkiens, *Systems Engineering with SysML/UML: Modeling, Analysis, Design*. Burlington, MA: Morgan Kaufmann, 2008.
[5] R. F. Paige, P. J. Brooke, and J. S. Ostroff, "Metamodel-based model conformance and multiview consistency checking," *ACM Trans. Softw. Eng. Methodol.*, vol. 16, no. 3, pp. B1–B49, 2007.
[6] H. Malgouyres and G. Motet, "A UML model consistency verification approach based on meta-modeling formalization," in *Proc. ACM Symp. Appl. Comput.*, 2006, pp. 1804–1809.
[7] D. Allaki, M. Dahchour, and A. Ennouaary, "A new taxonomy of inconsistencies in UML models with their detection methods for better MDE," *Int. J. Comput. Sci. Appl.*, vol. 12, no. 1, pp. 48–65, 2015.
[8] R. S. Bashir, S. P. Lee, and S. U. R. Khan, "UML models consistency management: Guidelines for software quality manager," *Int. J. Inf. Manage.*, vol. 36, pp. 883–899, 2016.
[9] K.-P. Lin and W. S. Chao, "The structure-behavior coalescence approach for systems modeling," *IEEE Access*, vol. 7, pp. 8609–8620, 2019.
[10] N. Przigoda, R. Wille, and R. Drechsler, "Analyzing inconsistencies in UML/OCL models," *J. Circuits, Syst. Comput.*, vol. 25, no. 3, pp. 1640021:1–1640021:21, 2016.
[11] OMG, *Semantics of a Foundational Subset for Executable UML Models (fUML)*. Object Management Group, Needham, MA, 2013.
[12] OMG, *Action Language for Foundational UML (Alf)*. Object Management Group, Needham, MA, 2013.
[13] ISO TC-184 (Technical Committee on Industrial Automation Systems and Integration), ISO 18629 Process Specification Languages (PSL), 2006.
[14] M. S. Fisher, *Software Verification and Validation: An Engineering and Scientific Approach*, 2007th Ed., New York, NY, USA: Springer, 2007.
[15] P. Sujatha, G. Vijaya Sankar, A. Sarvottama Rao, and T. Satyanarayana, "The role of software verification and validation in software development process," *IETE Tech. Rev.*, vol. 18, no. 1, pp. 23–26, 2015.

[16] W. S. Chao and S.-P. Sun, *Using Operation-Based Multi-Queue SBC Process Algebra as a Metamodel for UML: Toward a Unified View of the System*, Independently published, 2019.
[17] D. Rosenberg, M. Stephens, and M. Collins-Cope, *Agile Development with ICONIX Process*, New York, NY, USA: Apress, 205.
[18] D. Rosenberg and M. Stephens, *Use Case Driven Object Modeling with UML: Theory and Practice*, New York, NY, USA: Apress, 2013.
[19] C. Keating, R. Rogers, R. Unal, and D. Dryer, "System of systems engineering," *Eng. Manage. J.*, vol. 15, no. 3, 2015, Art. no. 36.
[20] C. B. Nielsen, P. G. Larsen, J. Fitzgerald, J. Woodcock, and J. Peleska, "Systems of systems engineering," *ACM Comput. Surv.*, vol. 48, no. 2, pp. 1–41, 2015.

Wei-Ming Ma (Member, IEEE) was born in Taipei, Taiwan. He received the master's degree in hydrographic sciences from the Naval Postgraduate School, CA, USA, in 1988 and the Ph.D. degree in physical oceanography from the Florida Institute of Technology, FL, USA, in 1997.

He was an Assistant Professor from 2002 to 2014. Since 2016, he has been an Associate Professor with the Department of Information Management, Cheng-Shiu University, Kaohsiung, Taiwan. His research interests include SBC architecture, enterprise architecture, information security, digital forensics, and AR/VR multimedia design.

William S. Chao was born in 1954 in Taiwan and received the Ph.D. degree in information science from the University of Alabama, Birmingham, AL, USA, in 1988.

He worked as a Computer Scientist with GE Research and Development Center, from 1988 to 1991 and has been teaching with National Sun Yat-Sen University, Taiwan, since 1992. His research interests include systems architecture, hardware architecture, software architecture, and enterprise architecture.

Dr. Chao is a member of the Association of Enterprise Architects Taiwan Chapter and also a member of the Chinese Association of Enterprise Architects.