IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 26, 2018, accepted January 2, 2019, date of publication January 7, 2019, date of current version January 23, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2891067

The Structure-Behavior Coalescence Approach for Systems Modeling

KENG-PEI LIN, (Member, IEEE), AND WILLIAM S. CHAO
Department of Information Management, National Sun Yat-sen University, Kaohsiung 80424, Taiwan
Corresponding author: Keng-Pei Lin (kplin@mis.nsysu.edu.tw)

This work was supported by the Ministry of Science and Technology, Taiwan, under Grant 106-2410-H-110-019-MY2.

ABSTRACT The systems modeling technique is used for developing an abstract model to help analyze and understand the functionality of a system. In this paper, we introduce a new approach for general-purpose systems modeling called structure-behavior coalescence (SBC), which supports the integration of modeling both structural and behavioral aspects of a system in a single diagram. Process algebra is utilized to describe the interactions between the components of the system. SBC combines graphical notation and process algebra to provide formalism and an intuitive representation in the development of the conceptual model. The SBC model can clearly describe the interactions between system components and precisely specify the execution order of an entire system via algebraic operations.

INDEX TERMS Systems modeling, process algebra, systems architecture, systems engineering.

I. INTRODUCTION

The need for systems modeling arises from the large and complex nature of real-world systems. Systems modeling is for conceptually developing and reasoning about the functions, data, structure, and behavior of systems, where a system model describes and represents all these various aspects in graphical or textual ways.

Various modeling techniques and languages have been developed for different purposes [1]–[6]. In the area of information systems design, the entity-relationship (ER) model [1] is a popular data modeling technique for specifying the relationships between entity types, and the Unified Modeling Language (UML) [2]–[4] is used for the visualization of object-oriented software designs.

General systems modeling languages include the Object-Process Methodology (OPM) [6]–[9] and the Systems Modeling Language (SysML) [4]–[6]. The OPM models both the structure and behavior components of a system in a single diagram. The SysML, which provides multiple views of a system with various diagrams, is an extension of the software-centric UML for general systems engineering and is considered to be more expressive for details [10].

The unification to a single diagram of OPM helps to avoid the model multiplicity problem caused by separating the object model (system structure) from the process model (system behavior) [11]. OPM provides an equivalent textual description of the OPM system diagram. The diagram provides an overall picture of the system, while the textual description helps to understand the details of the system. However, the textual description of OPM consists of human-readable English sentences but is not executable and lacks strict formalism, which means that automation of system modeling and simulation is difficult in OPM.

We are in favor of integrating objects with processes into a single diagram such as OPM and proposing the structure-behavior coalescence (SBC) approach, which leverages the mechanisms of *process algebra* [12]–[15]. SBC utilizes process algebra to model the behavior of systems, and the process algebra provides a mathematical structure for performing computations on processes with operators based on axioms [16]. Featuring process algebra, SBC enables the equational reasoning of system behavior and the description and specification of systems via algebra.

The main difference between SBC and OPM is that SBC uses only one semantic notation, i.e., the interaction between caller and callee, with process algebra to describe the executions of the system, whereas the OPM uses several semantic entities to connect processes to processes, processes to objects, and objects to objects with control flow and object states and conditions to specify the execution order. Algebra can be a representationally efficient way to reason about systems since the operands can incorporate the information of algebraic expressions and operations can be written in formal expressions with the associative or commutative formal

properties. SBC leverages process algebra to clearly describe the interactions in systems with simple and intuitive algebraic operations to reason about complex systems.

System behavior is composed of concurrent executions of several processes, which influence each other by exchanging messages. Process algebra is a diverse family of methods for formally modeling concurrent systems with high-level descriptions of synchronizations, communications, and interactions between a collection of processes. The SBC process algebra is based on the calculus of communicating systems (CCS) process algebra [13] due to its simplicity and intuitive representation of operations and the potential of extending to π-calculus, which provides process mobility [14]. The system behavior described by process algebra can be analyzed by equational reasoning and is executable algebraically. SBC capitalizes on the process algebra to provide formalism in the system modeling; hence, the SBC model is suited for high-level modeling as well as verification and simulation of complex systems. Early prototyping is possible for analyses by simulating the behavior under certain circumstances, and the verification can determine whether the system satisfies certain properties or displays the desired behavior.

The rest of this paper is organized as follows: Section II reviews the related studies. Section III develops the SBC approach for systems modeling. In Section IV, we present the application of SBC systems modeling and compare it with that of OPM and SysML. Section V gives the conclusion and discusses the limitations of SBC.

II. LITERATURE REVIEW

The notion of systems has been applied in various fields, such as systems theory [17], systems analysis and design [18], [19], and systems engineering [20], [21].

Systems thinking is the core principle of the systems engineering. The systems analyses of systems models reflect the systems thinking. Model-based systems engineering is the use of a formal model to reason about a problem [22], [23]. System models can be utilized to describe existing systems, to prescribe future systems, and to reason about or analyze a system.

The OPM [7], [8] is a general-purpose systems modeling language that integrates the structural and behavioral aspects of a system into a single diagram. The OPM represents the system as equivalent graphics and text. The graphical part, called the Object-Process Diagram, visualizes the system with notations of objects, states of objects, processes, and links by plotting a single diagram. The equivalent textual description, called the Object-Process Language, provides a clear way to investigate the details of the system. In OPM, the objects are static with optional quantitative or qualitative properties to define the states of objects, and the processes are dynamic, which can transform objects, for example, creating, destroying, or changing the states of objects. The OPM has a MATLAB extension that can be used to add computational ability to the OPM conceptual model [24]. Two approaches exist: the first is to represent the entire OPM model in MATLAB code, while the second is to replace the computation parts of processes in the OPM with either MATLAB functions or MATLAB Simulink models. These extensions enhance the computational ability of the OPM conceptual model. The integrated modeling approach of OPM helps to mitigate the model multiplicity problem and prevents the potential inconsistency that may occur in multi-diagram environments [8], [11]. Multi-diagram approaches such as SysML and UML utilize separate diagrams to describe different aspects of a system. A reader needs to simultaneously refer to various diagrams and fuse the diagrams mentally to comprehend the system and its operations. Our SBC method is also a single-diagram approach and therefore, like OPM, has the advantage of the inherent integration of structure and behavior. However, in OPM models, the sequence of executing processes is represented only implicitly by the control flow. By contrast, the SBC utilizes process algebra to explicitly and clearly specify the execution order of the entire system.

UML [2], [3] is an Object Management Group standard for the visualization, description, and modeling of object-oriented software system development. UML consists of standardized graphical notations for creating an abstract model of a system. UML uses a collection of diagrams to represent different aspects of a system. The SysML is an extension of a subset of UML 2.0 diagrams for general-purpose systems engineering [4]–[6]. SysML has nine diagrams for representing different aspects of systems. The SysML diagrams are categorized as structural or behavioral, and each category has four diagrams to reflect the various aspects of a system. The structural diagrams include the block definition diagram, internal block diagram, package diagram, and parametric diagram, and the behavioral diagrams include the use case diagram, activity diagram, sequence diagram, and state machine diagram. A special diagram in SysML is the requirement diagram, which supplies the text-based requirements of a system. The requirement diagram can be shown in a graphical, tabular, or tree structure format. Because requirements can appear in other diagrams to illustrate the relationships with other modeling elements, the requirement diagram provides a way to better integrate the system requirements with other diagrams.

Since SysML is a multi-view approach based on UML, there may be inconsistencies between different diagrams. To ensure and check the consistency, a metamodel [25], which defines the syntax of a modeling language, provides a unifying framework for defining consistency rules to impose constraints on components [5], [25], [26]. Consistency rules are usually defined in the object constraint language (OCL) [27] and the action language for foundational UML (Alf) [28], [29] to check for conflicting syntax or semantics between the diagrams. Our SBC adopts an OPM-like single diagram approach to prevent the multi-diagram inconsistencies, and the process algebra-based interactions in SBC enforce strict formalism on the model.

The Universal Systems Language (USL) [30] is a formal modeling language for the design of software and complex systems and can provide formal semantics to support SysML specifications [31]. The USL was developed based on axioms of a general systems theory to provide formal foundations. In the USL, a system is defined as function maps (FMaps) representing dynamic actions and type maps (TMaps) representing static objects, with all maps being defined in terms of three control structures: ''join'', ''include'', and ''or''. However, use of these primitive structures can be cumbersome when designing the structures of complex systems. SBC uses only one kind of semantic link, i.e., the interaction, to connect components and represent processes, which is succinct and intuitive, and the control flow of SBC is clearly specified by the process algebra.

The Object-Process Network (OPN) [32] is an executable modeling language for systems architecting. Algebraic operations are introduced into the single-diagram representation of OPM to represent and manipulate the simulation models. OPN enforces bipartite graph formalism, allowing only direct connections from objects to processes and vice versa, which helps to implement execution engines based on graphical computing models. However, the Petri net-like connections between objects and processes in OPN can be difficult in conceptual modeling. By contrast, SBC utilizes simple but intuitive process algebra to model the processes and connect objects, precisely specifying the execution order and providing a clear and efficient representation for reasoning about a complex system.

Process algebra offers algebraic rules for the specification of processes and allows formal reasoning about and analysis of the system behavior. In addition to the original goal of describing concurrent systems, process algebra has also been utilized to reason about cryptographic protocols [33], business processes [34], and systems biology [35]. SBC capitalizes on the expressiveness and equational reasoning properties of process algebra to develop a system modeling approach that clearly describes the interactions among system components with simple algebraic operations.

III. THE STRUCTURE-BEHAVIOR COALESCENCE APPROACH

In this section, we introduce the SBC approach and its rationale. We first describe the interactions, which convey both the structural and behavioral information of the system. Then, we describe the SBC process algebra through which the interactions are formulated as equations in SBC process algebra. Finally, we construct the SBC syntax and define the transition semantics of the SBC process algebra. An example of SBC modeling is presented in the next section.

A. OPERATION-BASED VALUE-PASSING INTERACTIONS

In SBC, all components are linked by interactions. The interaction is the only semantic entity of the SBC approach and is formalized as an operation-based method for passing values.

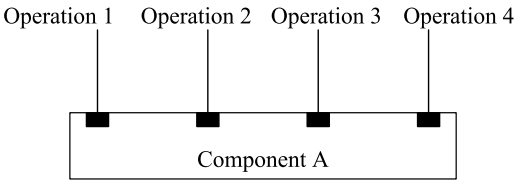

FIGURE 1. A component may have several operations.

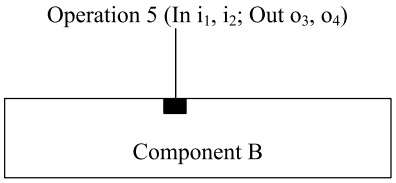

FIGURE 2. An operation may contain several input/output parameters.

The component is a physical or informatical entity. The component can also be a department or personnel responsible for specific functions. An operation equipped with a component represents a procedure, a method, or a function of the component. A component may be associated with several operations, as shown in Fig. 1. An operation may contain several input parameters (e.g., i_1, i_2) and output parameters (e.g., o_3, o_4), as shown in Fig. 2. These input and output parameters may represent matter, energy, data, information or messages.

An operation formula is used to completely describe an operation. An operation formula consists of the operation name, input parameters (e.g., $i_1, i_2, \ldots, i_m$), and output parameters (e.g., $o_1, o_2, \ldots, o_n$).

To define the process algebra, we start with a set of interactions whose purpose is to offer the basic form of synchronization and communication between agents. An interaction represents an indivisible and instantaneous handshake or rendezvous between two agents [12]–[15]. In the operation-based value-passing interaction approach shown in Fig. 3, the caller agent (either an external environment actor or component) interacts with the callee agent (component) through the operation call or operation return interaction. The solid line indicates an operation call from the caller to the callee, and the dashed line indicates an operation return from the callee to the caller. In the figure, `getPastDueBalance(In studentId)` is an operation call formula and `getPastDueBalance(Out PastDueBalance)` is an operation return formula. The operation call formula and its corresponding operation return formula can be merged into an operation formula.

The external environment uses a ''type 1 interaction'' to interact with a component. We formally describe an operation-based value-passing type 1 interaction as a 4-tuple `Type 1 Interaction = <operation call or return, actor, operation call or return formula, callee component>`, where `operation call or return` represents an `OPERATION CALL` or

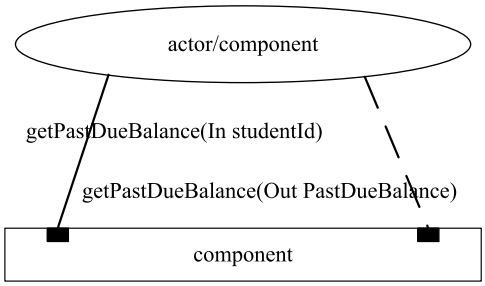

FIGURE 3. Operation-based value-passing interactions.

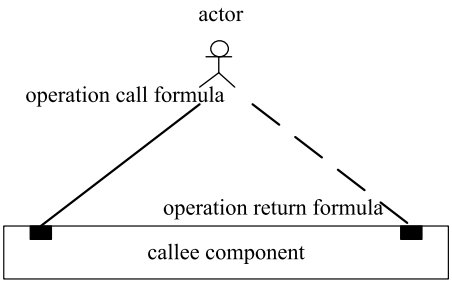

FIGURE 4. Formal description of a type 1 interaction.

`OPERATION RETURN` tag, `actor` represents the name of the external environment's actor, `operation call or return formula` represents an operation call or operation return formula and `callee component` represents the name of the callee component, as shown in Fig. 4.

Two components use a "type 2 interaction" to interact with each other. We formally describe an operation-based value-passing type 2 interaction as a 4-tuple `Type 2 Interaction = <operation call or return, caller component, operation call or return formula, callee component>`, where `operation call or return` represents an `OPERATION CALL` or `OPERATION RETURN` tag, `caller component` represents the name of the caller component, `operation call or return formula` represents an operation call or operation return formula and `callee component` represents the name of the callee component, as shown in Fig. 5.

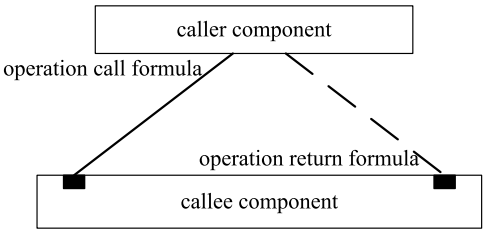

FIGURE 5. Formal description of a type 2 interaction.

B. MATHEMATICS OF THE SBC PROCESS ALGEBRA

A process consists of a sequence of interactions between components of a system. Therefore, in addition to the interactions, we need operators to constitute new processes [12]–[15].

TABLE 1. Entities of SBC process algebra.

Entity set	Entity name	Type of entity
G	$g_1, g_2, \ldots$	type 1 interactions
V	$v_1, v_2, \ldots$	type 2 interactions
Δ	$a_1, a_2, \ldots$	type 1 or 2 interactions
Ψ	$E_1, E_2, \ldots$	process expressions
Φ	$A_1, A_2, \ldots$	process constants
χ	$X_1, X_2, \ldots$	process variables
I	i, j	index set

We first define the entities used in the SBC process algebra. We assume a possibly infinite set of operation call or operation return formulas. As shown in Table 1, we let G be the set of type 1 interactions, ranged over by g.

We let V be the set of type 2 interactions, ranged over by v. We let Δ be the set of type 1 or 2 interactions, ranged over by a.

Process expressions describe processes in mathematical notation. A process expression represents a state of the system, incorporating both structure and behavior information. A sequential process indicates state transitions, and processes can be combined to make larger processes. We let Ψ be the set of process expressions, ranged over by E.

Process constants are used to name processes. For example, $A \stackrel{\text{def}}{=} E$ means that E is the body of the defined constant A. We let Φ be the set of process constants, ranged over by A. A process variable can be instantiated by a process. We let χ be the set of process variables, ranged over by X.

Then, we introduce the operators of SBC process algebra for constituting a new process. The operators include sequential composition of processes, summation of processes, parallel composition of processes, recursive definition of a process, and the null process, as described below.

1) SEQUENTIAL COMPOSITION OF PROCESSES

Sequential composition is known from other models of process executions. Sometimes process executions must be temporally ordered. For example, it might be appropriate to describe algorithms such as execute the E_1 process first and then execute the E_2 process later. Sequential composition can be used for such purposes. The sequential composition of two processes E_1 and E_2, generally written as $E_1 \bullet E_2$, indicates that the executions in E_1 and E_2 must proceed sequentially.

2) SUMMATION OF PROCESSES

The binary operator "+", summation, combines two process expressions as alternatives; i.e., it represents the choice between alternative process expressions. For example, the process E_1+E_2 can proceed either as the process E_1 or the process E_2. As soon as the first interaction occurs, the other is discarded.

3) PARALLEL COMPOSITION OF PROCESSES

Parallel composition of two processes E_1 and E_2, written as $E_1||E_2$, distinguishes the process algebra from sequential

models of process executions. Parallel composition permits the executions in E_1 and E_2 to proceed independently and concurrently.

4) RECURSIVE DEFINITION OF A PROCESS

The operators presented so far depict only finite interactions and are therefore insufficient for full computability, which contains non-terminating (or looping) behavior. Recursion is the operator that allows finite descriptions of infinite behavior. For example, **fix**$(X = E)$ is understood as the abbreviation of the recursive definition of an infinite behavior denoted by the X process variable. Here, we utilize the fixed-point combinator to give recursive definitions [36]. **fix** is a higher-order function that returns the fixed point of the function. **fix**$(X = E)$ means that E is the expression of the recursive process X.

5) NULL PROCESS

Process algebra usually also includes a null process, denoted as STOP, which has no interaction point. This process is completely inactive, and its unique intention is to act as the inductive anchor on top of which some processes can be generated.

C. TRANSITIONAL SEMANTICS OF THE SBC PROCESS ALGEBRA

To give meaning to the SBC process algebra, we shall use the following transition system

$$(\Psi, \Delta, \{\xrightarrow{a} : a \in \Delta\})$$

which consists of a set Ψ of process expressions, a set Δ of transition type 1 or 2 interactions, and a transition relation $\xrightarrow{a} \subseteq \Psi \times \Psi$ for each $a \in \Delta$. The semantics for Ψ consist of the transition rules of each transition relation over Ψ. These transition rules follow the structure of expressions.

$$\text{Prefix} \quad \frac{}{a \bullet E \xrightarrow{a} E}$$

$$\text{Sum}_j \quad \frac{E_j \xrightarrow{a} E'_j}{\sum_{i \in I} E_i \xrightarrow{a} E'_j} \ (j \in I)$$

$$\text{Recursion} \quad \frac{E\{\mathbf{fix}(X=E)/X\} \xrightarrow{a} E'}{\mathbf{fix}(X=E) \xrightarrow{a} E'}$$

$$\text{Constant} \quad \frac{E \xrightarrow{a} E'}{A \xrightarrow{a} E'} \ (A \overset{\text{def}}{=} E)$$

FIGURE 6. Transition rules for SBC process algebra.

Fig. 6 gives the complete set of transition rules: Prefix, Summation, Recursion and Constant.

The rule for Prefix can be read as follows: Under any circumstances, we always infer $a \bullet E \xrightarrow{a} E$, i.e., an expression with an interaction prefixed to it will use this interaction to accomplish the transition.

The rule for Summation can be read as follows: If any one summand E_j of the sum $\sum_{i \in I} E_i$ has an interaction, then the whole sum also has that interaction. Finite Summation, which is sufficient for many practical purposes, can be presented in a more convenient form. If $I = \{1, 2\}$, then we obtain two rules for $E_1 + E_2$ by setting $j = 1, 2$:

$$\frac{E_1 \xrightarrow{a} E'_1}{E_1 + E_2 \xrightarrow{a} E'_1} \quad \frac{E_2 \xrightarrow{a} E'_2}{E_1 + E_2 \xrightarrow{a} E'_2}.$$

Additionally, we define $STOP \overset{\text{def}}{=} \sum_{i \in \emptyset} E_i$. Because $I = \emptyset$ in this case, there is no rule for the null process $STOP$, which reflects the fact that $STOP$ has no transitions.

The rule for Recursion can be read as follows: Any interaction, which may be inferred for the **fix** expression expanded once by substituting itself for its bound variable, can be inferred for the **fix** expression itself.

The rule for Constant can be read as follows: Each constant has the same transitions as its defining expression.

D. LANGUAGE CONSTRUCTS OF THE SBC PROCESS ALGEBRA

We now define the SBC syntax for describing the whole system in Backus-Naur form (BNF) [37]. BNF is a notation technique for describing the syntax of languages used in programming languages, document formats, communication protocols, etc. In the SBC approach, the syntax of the process algebra is defined with the following BNF grammar.

(1)<System>
::="**fix**("<Proces Variable>"="
$\sum_{i \in I}$<IFD$_i$>")"

(2)<IFD>::= <Type 1 Interaction>
{"•"<Type 1 or 2 Interaction>}
"•"<Process Variable>

(3)<Type 1 or 2 Interaction>
::=<Type 1 Interaction>|
<Type 2 Interaction>

Rule 1 states that the recursion (i.e., **fix**) of summation (i.e., $\sum_{i \in I}$) of one or more interaction flow diagrams (i.e., IFD$_i$) defines the SBC process expression of a system. Rule 2 states that an interaction flow diagram is defined by a type 1 interaction followed by zero or more type 1 or 2 interactions. Rule 3 states that a type 1 or 2 interaction is either a type 1 interaction or a type 2 interaction.

In the SBC process algebra, the process of a system is defined as **fix**$(X = \sum_{i \in I}(IFD_i \bullet X))$, and the process of IFD_i is defined as $\bullet_{j \in J} a_{ij}$, where $a_{i1} = g_{i1}$ for all $i \in I$. To combine

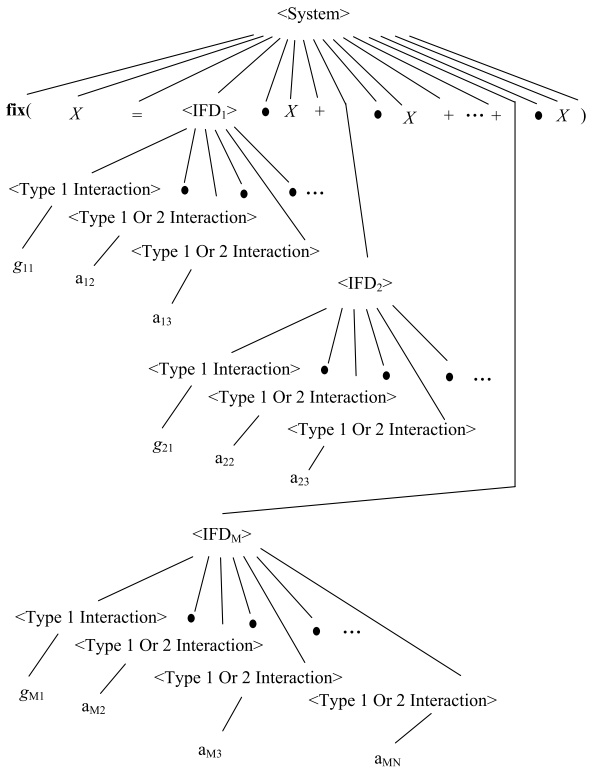

FIGURE 7. SBC Backus-Naur form tree of a system.

these definitions, we use the SBC process algebra to formally define a system as

$$\mathbf{fix}(X = g_{11} \bullet a_{12} \bullet a_{13} \bullet \cdots \bullet X + g_{21} \bullet a_{22} \bullet a_{23} \bullet \cdots \bullet X + \cdots + g_{M1} \bullet a_{M2} \bullet a_{M3} \bullet \cdots \bullet a_{MN} \bullet X).$$

We utilize the BNF tree to illustrate the syntactic structure of the entity sequence that constitutes the SBC process. The SBC process algebra BNF tree of a system is shown in Fig. 7.

The SBC process algebra consists of two parts. The first part models the interactions that occur in the process of a system, and the second part models the execution order of the entire system. The interactions provide the basic form of communication between agents in the SBC process algebra. The agent may be an external environment actor or a component. All the interactions of the SBC process of a system are shown in Fig. 8.

Based on the SBC transitional semantics, whenever $E \xrightarrow{a_1} \cdots \xrightarrow{a_n} E'$, we say that $(a_1 \cdots a_n, E')$ is a derivative of E. The derivatives of a process expression E can conveniently be collected into the SBC transition graph of E. We use the SBC transition graph to define the execution of the entire system, as shown in Fig. 9. In the SBC transition graph, a process expression is represented by a labeled circle, edges are used to represent the "transition" between two process expressions, and the starting process expression is usually represented by an arrow with no origin pointing to the process expression.

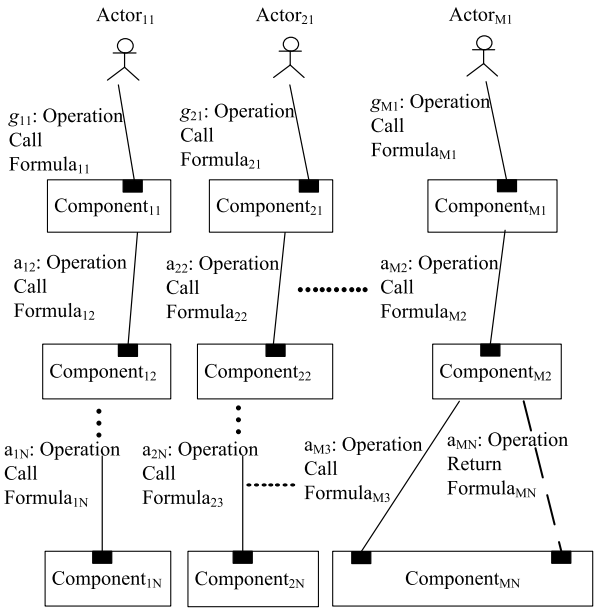

FIGURE 8. Interactions of the SBC process of a system.

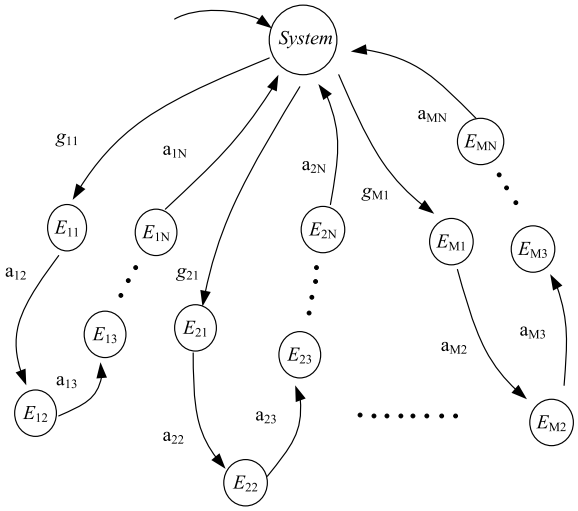

FIGURE 9. SBC transition graph of a system.

E. SIMULATION OF THE SBC PROCESS ALGEBRA

We implement the simulator of SBC models to execute the SBC process algebra. The SBC simulation adopts a single-queue interaction scheduling algorithm. Only one queen is present in the SBC simulation, as shown in Fig. 10. When the execution of an interaction flow diagram is ready, the to-be-executed interactions, such as $g_{11}, a_{12}, \ldots, a_{MN}$, are added one-by-one to the end of the SBC simulation queue. *Ready-Head* points to the interaction at the head of the queue. The SBC simulation scheduling algorithm picks only the to-be-executed interaction at the head of that queue. When the queue becomes empty, the idle routine is executed.

IV. THE APPLICATION OF SBC SYSTEMS MODELING

We illustrate the application of SBC systems modeling with a vending machine example. A vending machine is an

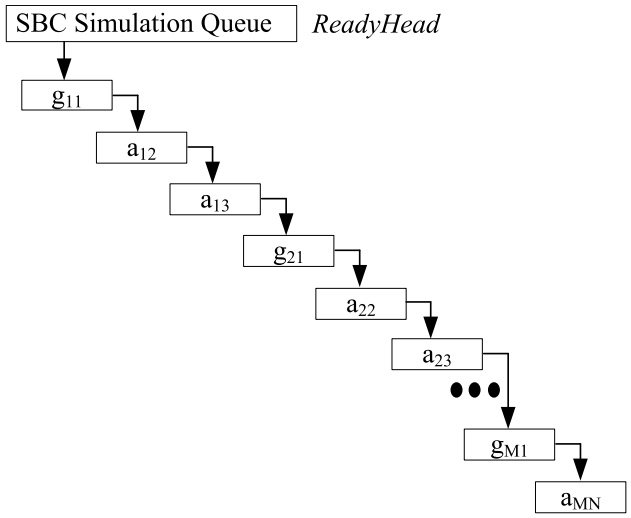

FIGURE 10. The SBC simulation.

automated machine that sells products to customers. A customer inserts enough money for a product and selects a specific product; then, the vending machine dispenses the selected product and dispenses change if necessary. A vending machine is a rather complex system with several features. Customers interact with the coin receptacle and buttons of the vending machine, and the vendor needs to refill the vending products and change. To create a system model of a vending machine, we begin by describing the operational scenario of the vending machine.

The vending machine accepts coins or credit from the customer in payment for their purchase. The vending machine returns the customer's payment if he or she decides not to make a product selection. The vending machine accepts the customer's product selection. Once the customer makes a selection, the vending machine dispenses the product to the customer. To ensure that most vending products are available for customers to purchase, the vendor regularly checks the product store to see if any product needs to be refilled. The vendor refills the product store as needed. The vendor regularly checks and refills the coin store as needed to ensure that there are deposited coins available for change.

All the interactions of the vending machine SBC process are shown in Fig. 11. The SBC process algebra for modeling the vending machine, E_{000}, is defined as $\mathbf{fix}(X = g_{11} \bullet v_{12} \bullet v_{13} \bullet v_{14} \bullet X + g_{21} \bullet v_{22} \bullet v_{23} \bullet v_{24} \bullet g_{25} \bullet v_{14} \bullet X + g_{31} \bullet v_{32} \bullet v_{33} \bullet v_{34} \bullet g_{35} \bullet v_{23} \bullet v_{24} \bullet g_{25} \bullet v_{14} \bullet X + g_{41} \bullet X + g_{51} \bullet X)$.

The BNF tree of the vending machine SBC process algebra is shown in Fig. 12, and the SBC transition graph is shown in Fig. 13, which displays the execution order for the vending machine. In the SBC transition graph of the vending machine, processes E_{000}, E_{001}, E_{002}, E_{003}, E_{004}, E_{005}, E_{006}, E_{007}, E_{008}, E_{009}, E_{010}, E_{011}, E_{012}, E_{013}, E_{014}, E_{015}, and E_{016} are defined in Fig. 14.

We apply the SBC simulation algorithm to the SBC systems modeling of the vending machine. The scenario is as follows: When the customer puts in a 25-cent coin and selects

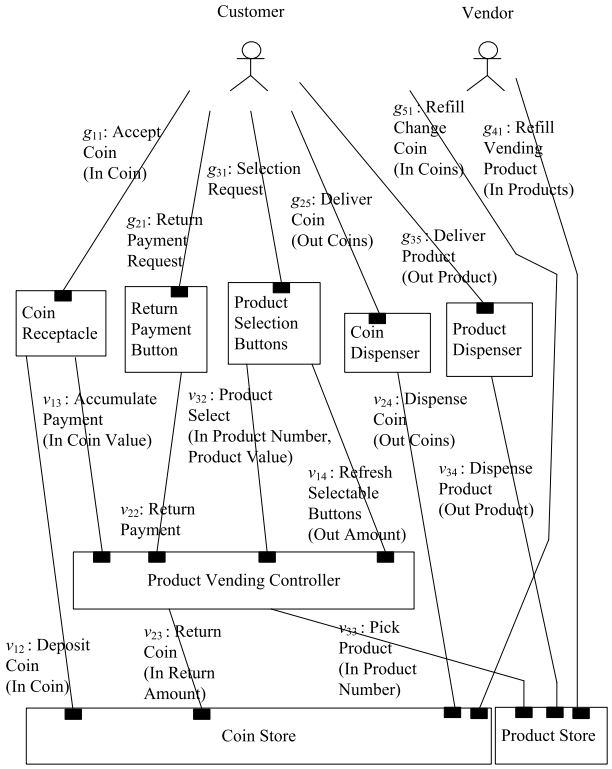

FIGURE 11. Interactions of the vending machine SBC process.

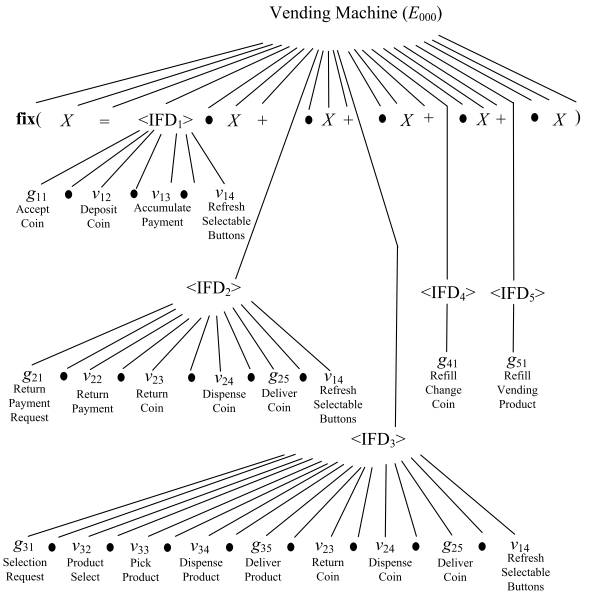

FIGURE 12. SBC Backus-Naur form tree of the vending machine.

a product selling for 20 cents, the machine will dispense the product and return 5 cents. The result of the SBC vending machine simulation is shown in Fig. 15. The execution order is the same as the transition sequence shown in the SBC transition graph of the vending machine in Fig. 13, i.e., $g_{11} \rightarrow v_{12} \rightarrow v_{13} \rightarrow v_{14} \rightarrow g_{31} \rightarrow v_{32} \rightarrow v_{33} \rightarrow v_{34} \rightarrow g_{35} \rightarrow v_{23} \rightarrow v_{24} \rightarrow g_{25} \rightarrow v_{14}$, which validates the correctness of the modeling.

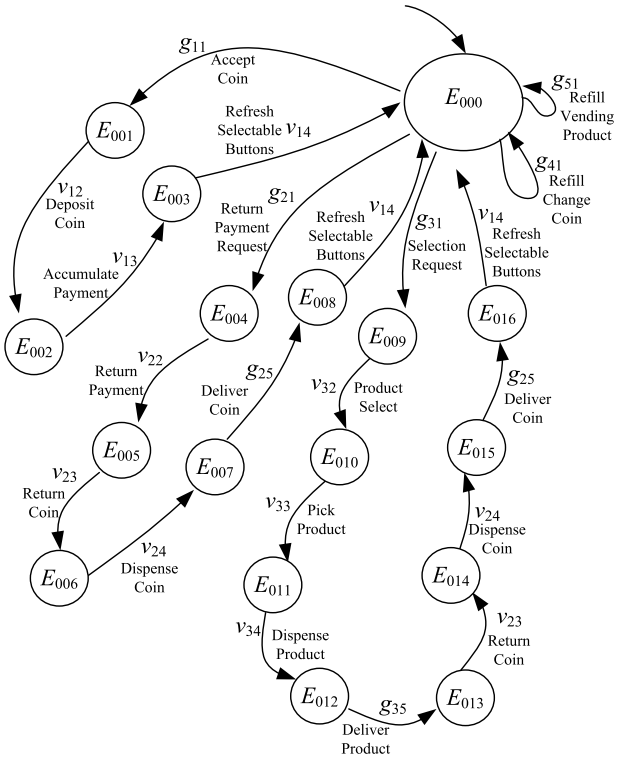

FIGURE 13. SBC transition graph of the vending machine.

$$
\begin{aligned}
E_{000} &\stackrel{\text{def}}{=} g_{11}(\text{Accept Coin})\bullet E_{001} + g_{21}(\text{Return Payment Request})\bullet E_{004} \\
&\quad + g_{31}(\text{Selection Request})\bullet E_{009} + g_{41}(\text{Refill Change Coin})\bullet E_{000} \\
&\quad + g_{51}(\text{Refill Vending Product})\bullet E_{000} \\
E_{001} &\stackrel{\text{def}}{=} v_{12}(\text{Deposit Coin})\bullet E_{002} \\
E_{002} &\stackrel{\text{def}}{=} v_{13}(\text{Accumulate Payment})\bullet E_{003} \\
E_{003} &\stackrel{\text{def}}{=} v_{14}(\text{Refresh Selectable Buttons})\bullet E_{000} \\
E_{004} &\stackrel{\text{def}}{=} v_{22}(\text{Return Payment})\bullet E_{005} \\
E_{005} &\stackrel{\text{def}}{=} v_{23}(\text{Return Coin})\bullet E_{006} \\
E_{006} &\stackrel{\text{def}}{=} v_{24}(\text{Dispense Coin})\bullet E_{007} \\
E_{007} &\stackrel{\text{def}}{=} g_{25}(\text{Deliver Coin})\bullet E_{008} \\
E_{008} &\stackrel{\text{def}}{=} v_{14}(\text{Refresh Selectable Buttons})\bullet E_{000} \\
E_{009} &\stackrel{\text{def}}{=} v_{32}(\text{Product Select})\bullet E_{010} \\
E_{010} &\stackrel{\text{def}}{=} v_{33}(\text{Pick Product})\bullet E_{011} \\
E_{011} &\stackrel{\text{def}}{=} v_{34}(\text{Dispense Product})\bullet E_{012} \\
E_{012} &\stackrel{\text{def}}{=} g_{35}(\text{Deliver Product})\bullet E_{013} \\
E_{013} &\stackrel{\text{def}}{=} v_{23}(\text{Return Coin})\bullet E_{014} \\
E_{014} &\stackrel{\text{def}}{=} v_{24}(\text{Dispense Coin})\bullet E_{015} \\
E_{015} &\stackrel{\text{def}}{=} g_{25}(\text{Deliver Coin})\bullet E_{016} \\
E_{016} &\stackrel{\text{def}}{=} v_{14}(\text{Refresh Selectable Buttons})\bullet E_{000}
\end{aligned}
$$

FIGURE 14. Definition of processes of the vending machine.

A. COMPARISON OF SBC, OPM AND SYSML MODELS OF THE VENDING MACHINE

To compare SBC with the most popular general-purpose systems modeling approaches, we consider the object process diagram (OPD) of the OPM systems modeling of the vending machine shown in Fig. 16 and the internal block diagram, activity diagram, and requirement diagram of SysML presented in Fig. 17, Fig. 18 and Fig. 19, respectively. SysML consists of nine types of diagrams. Due to space limitations,

```
g_11, Caller:Customer, Callee:Coin Receptacle
Operation:Accept Coin(In Coin:25-cent)
v_12, Caller:Coin Receptacle, Callee:Coin Store
Operation:Deposit Coin(In Coin:25-cent)
v_13, Caller:Coin Receptacle, Callee:Product Vending Controller
Operation:Accumulate Payment(In Coin Value:25)
v_14, Caller:Product Selection Buttons, Callee:Product Vending Controller
Operation:Refresh Selectable Buttons(Out Amount:25)
g_31, Caller:Customer, Callee:Product Selection Buttons
Operation:Selection Request
v_32, Caller:Product Selection Buttons, Callee:Product Vending Controller
Operation:Product Select(In Product Number:808, Product Value:20)
v_33, Caller:Product Vending Controller, Callee:Product Store
Operation:Pick Product(In Product Number:808)
v_34, Caller:Product Dispenser, Callee:Product Store
Operation:Dispense Product(Out Product:808-soda)
g_35, Caller:Customer, Callee:Product Dispenser
Operation:Deliver Product(Out Product:808-soda)
v_23, Caller:Product Vending Controller, Callee:Coin Store
Operation:Return Coin(In Return Amount:5)
v_24, Caller:Coin Dispenser, Callee:Coin Store
Operation:Dispense Coin(Out Coins:5-cent)
g_25, Caller:Customer, Callee:Coin Dispenser
Operation:Deliver Coin(Out Coins:5-cent)
v_14, Caller:Product Selection Buttons, Callee:Product Vending Controller
Operation:Refresh Selectable Buttons(Out Amount:0)
```

FIGURE 15. The result of the SBC vending machine simulation.

we show only three diagrams of the vending machine model. The activity diagram, which displays the work flow of the system, including the input, output, and control between actions, is one of the four SysML behavioral diagrams. The internal block diagram is one of the four SysML structural diagrams and describes the internal structure of a system in terms of ports, parts, and connectors [6].

In the OPM example of the vending machine, the objects are denoted by rectangles, and the processes are denoted by ellipses. The connecting line with a black circle at the end, which starts at an object and ends at a process, is the agent link with the corresponding reserved phrase 'handles'. The agent link denotes that the object is a human who enables the process. The instrument link with the reserved phrase 'requires' is a connecting line from an object to a process with a white circle at the end that indicates that a non-human enables or controls the process. The arrows with white triangular heads from processes to objects are the result links with the corresponding reserved phrase 'yields'. The result link indicates that the process affects the object. A portions of the equivalent textual descriptions, i.e., the object process language (OPL), of the vending machine are presented for the abovementioned three types of links: **Customer** handles **Accept Coin**. **Accept Coin** yields **Coin Receptacle**. **Deposit Coin** requires **Coin Receptacle**.

Comparison of the OPM systems modeling diagram of the vending machine in Fig. 16 with that of the SBC in Fig. 11 shows that the conceptual structures of the OPM and SBC models are similar. The main difference between the two approaches is the link: the SBC approach uses only the 'interaction' to link the caller agent to the callee agent and represents the processes as operation-based value-passing interactions.

The OPM is able to concisely describe both the structural relations and behavioral relations among the objects and processes with the various types of links. The SBC describes the interactions with process algebra. The vending machine SBC process algebra is graphically shown in the SBC BNF

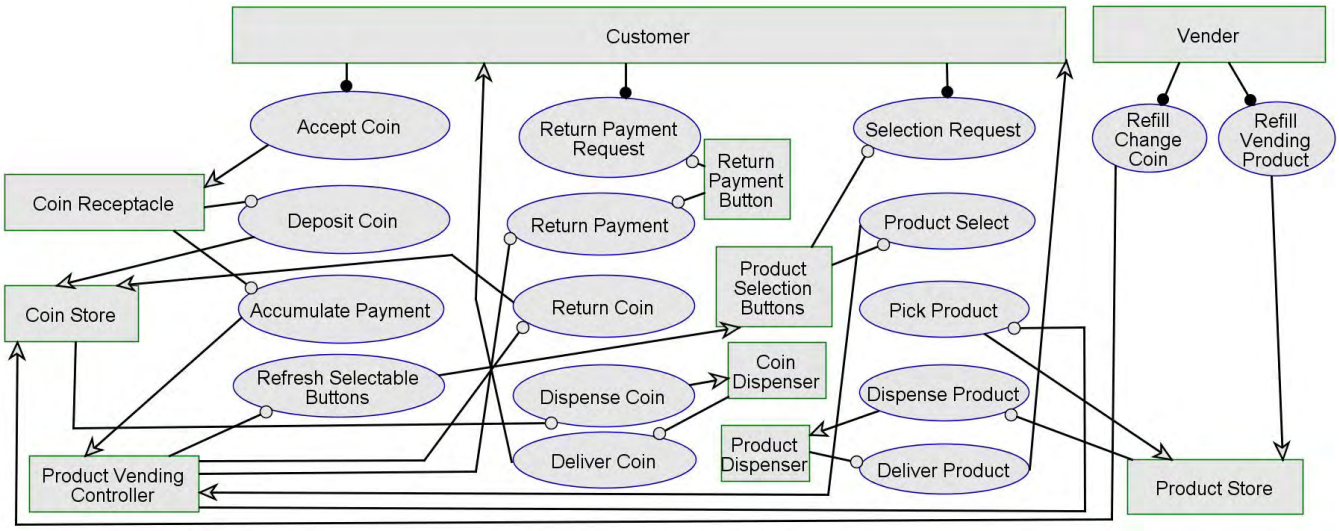

FIGURE 16. OPM systems modeling of the vending machine.

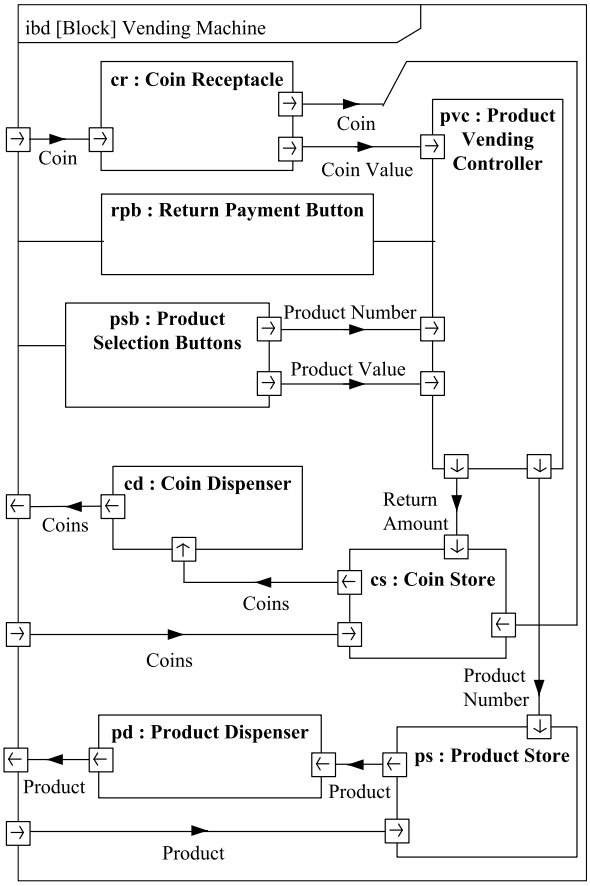

FIGURE 17. SysML internal block diagram of the vending machine.

tree in Fig. 12 and the SBC transition graph in Fig. 13. The SBC process algebra can clearly model the execution order of the system, i.e., the sequence, selection, and repetition of the processes. The SBC approach uses strict criteria to explicitly and precisely define the execution order of the entire system. By contrast, the OPM approach uses the control flow, timeline, conditions, and object states to model the execution order of the vending machine. By default, the timeline in the OPM model flows from top to bottom. Accordingly, the semantics of a process depicted above another process indicates that the process on top takes place prior to the one below it, unless the control flow indicates a different order. Generally, the top-to-bottom process arrangement in the OPM model denotes the default order of the execution. Since the control flow permits control structures, such as loops, the order of process execution in the OPM model determined by the control flow takes precedence over the order indicated by the timeline. Processes at the same height in the OPM can occur alternatively or simultaneously. Thus, **Accept Coin**, **Return Payment Request**, and **Selection Request** in Fig. 16 are alternative processes.

The control flow mechanism of the OPM does not clearly show the execution order of the processes. For example, in the vending machine, if the customer inserts more coins than required for the selected product, after completion of the process **Deliver Product**, the process **Return Coin** will occur. This flow is not explicitly shown in the OPM model of the vending machine since the process **Return Coin** and the corresponding subsequent processes are also carried out upon completion of **Return Payment Request** and are depicted in the timeline belonging to **Return Payment Request**. This deviation of the timeline is indicated by internal events.

A large and complex system may have many components, and a component may interact with several processes. It is difficult to prearrange all the processes in the top-to-bottom flow according to the execution order of processes. The other deviation of the timeline is the loop behavior. When a series of processes in response to a customer's action is completed, the control loops back to the customer. Since the OPM

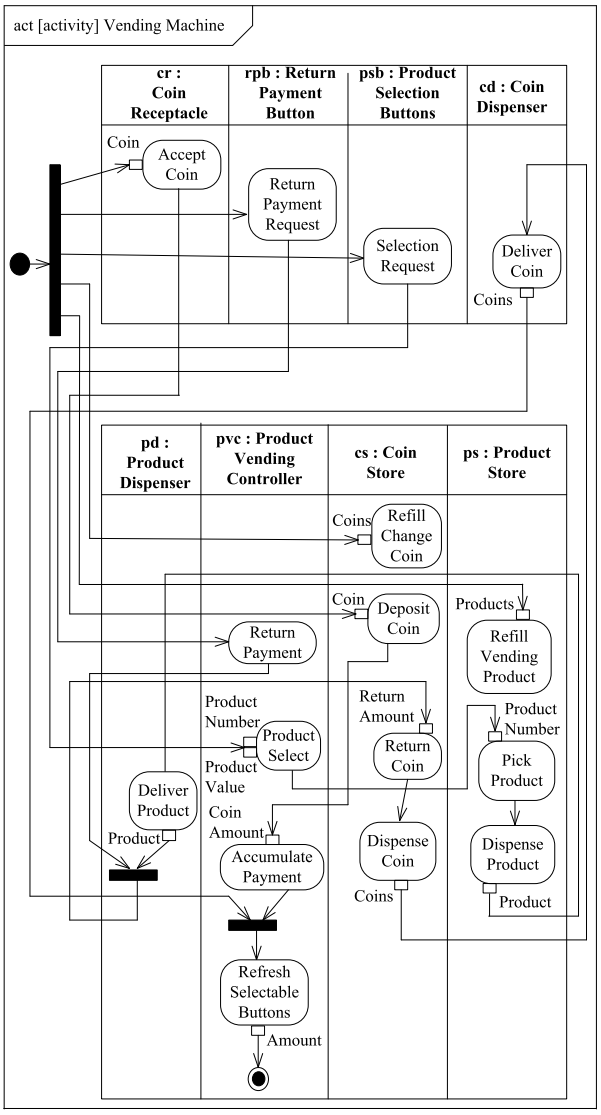

FIGURE 18. SysML activity diagram of the vending machine.

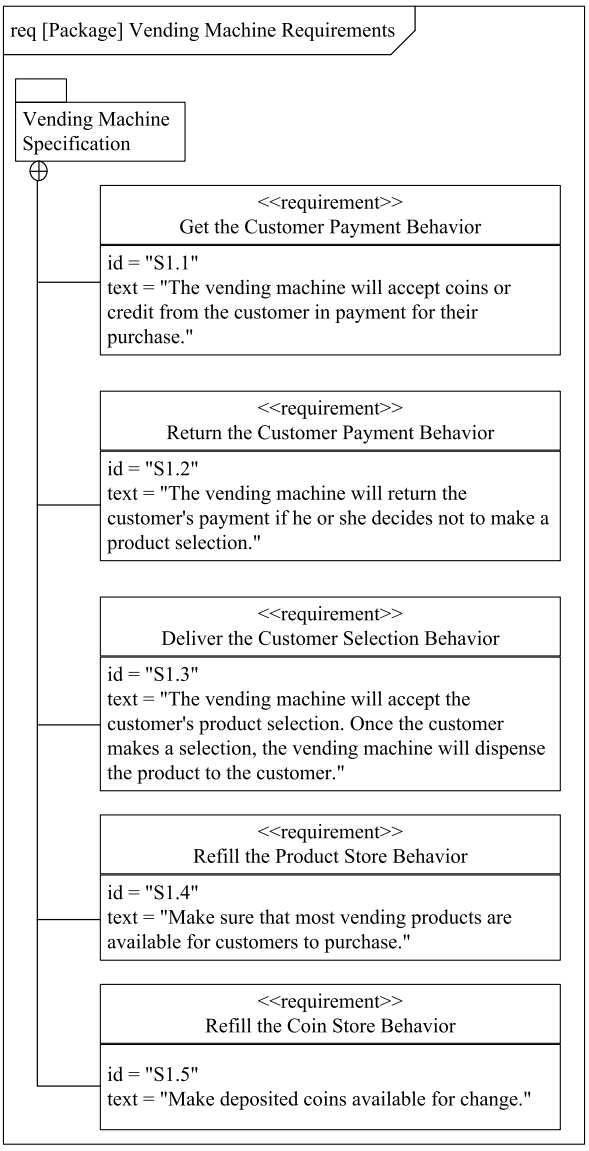

FIGURE 19. SysML requirement diagram of the vending machine.

timeline approach cannot describe the flow of the loop explicitly, it is also indicated by internal events.

In contrast, the SBC model describes the execution order in process expressions. The sequence of executing interactions strictly follows the process expressions, as well as the loop behavior, which is common in complex systems. By following the SBC process algebra, SBC can visualize the execution order in the transition graph; hence, the correctness of the execution order in the simulation is easily verifiable. Clearly, SBC outperforms the OPM in terms of accuracy and expressiveness in modeling the execution order of processes.

The SysML, like the OPM approach, also uses the control flow to indicate the execution order of the system. The SysML activity diagram of the vending machine is shown in Fig. 18, which describes the input, output, and control among the sequence of actions. SysML has a rich set of symbols to clearly define the sequence of workflows, and the requirement diagram of the SysML model can appear on other types of SysML diagrams for better integration of the system requirements with the elements of the model. The parametric diagram of the SysML model, which is not shown here, imposes property constraints.

Compared to SBC, the SysML model provides more detailed underlying information regarding the system. However, the presence of many different SysML diagrams requires analysts to exert greater effort to read them and is prone to inconsistency. For example, inconsistency exists between the internal block diagram in Fig. 17 and the activity diagram in Fig. 18. The block ''cr: Coin Receptacle'' has the ''Coin Value'' data stream to the block ''pvc: Product Vending Controller'' in the internal block diagram, but no ''Coin Value'' data stream exists in the activity diagram. The separation of structures and behaviors leads to difficulties in finding the inconsistencies between different diagrams, especially for a large and complex system. Consistency

checks are required through OCL or Alf to ensure that there are no inconsistencies between the diagrams. The SBC model is a single diagram, which could prevent potential inconsistencies between different diagrams. Furthermore, in contrast to the integration of SysML, which depends on a nonformal text-based requirement diagram, SBC enforces strict formalism on the integration based on process algebra.

V. CONCLUSIONS

In this paper, we propose the SBC systems modeling approach that takes advantage of the mechanisms of process algebra and integrates the structural and behavioral aspects of systems into a single diagram. The SBC systems modeling consists of two parts. The first part models the interactions that occur in the processes of a system, and the second part models the execution order of the system. The BNF is utilized to describe the syntax of the SBC approach. The SBC approach uses only one semantic entity, the interaction, to link the caller agent to the callee agent and uses the transitional semantic and transition graph for the execution specification of the system. This specification explicitly and precisely represents both the interactions between system components and the execution order of the entire system.

So far, the SBC process algebra proposed in this paper cannot model real-time systems, which is a limitation of the current SBC approach. Real-time systems [38] describe software and hardware systems subject to a ''real-time constraint'', for example, from events to systems responses. Events may occur at regular or irregular times, and the response must occur at predictably exact times. Real-time systems need to guarantee timely responses within specified timing constraints. To enable the SBC approach to model real-time systems, we need to extend the SBC process algebra with a timing mark or timing constraint specification, where a timing mark is an expression for the time at which an event occurs and a timing constraint is a semantic statement about the absolute or relative value of time. This extension will be addressed in our future work on the SBC approach.

REFERENCES

[1] P. P.-S. Chen, ''The entity-relationship model—Toward a unified view of data,'' *ACM Trans. Database Syst.*, vol. 1, no. 1, pp. 9–36, 1976.
[2] M. R. Blaha and J. R. Rumbaugh, *Object-Oriented Modeling and Design With UML*. London, U.K.: Pearson, 2004.
[3] J. Arlow and I. Neustadt, *UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design*. Reading, MA, USA: Addison-Wesley, 2005.
[4] T. Weilkiens, *Systems Engineering With SysML/UML: Modeling, Analysis, Design*. San Mateo, CA, USA: Morgan Kaufmann, 2008.
[5] S. Friedenthal, A. Moore, and R. Steiner, *A Practical Guide to SysML: The Systems Modeling Language*. San Mateo, CA, USA: Morgan Kaufmann, 2014.
[6] D. Dori, *Model-Based Systems Engineering With OPM and SysML*. New York, NY, USA: Springer-Verlag, 2016.
[7] D. Dori, ''Object-process analysis: Maintaining the balance between system structure and behaviour,'' *J. Logic Comput.*, vol. 5, no. 2, pp. 227–249, 1995.
[8] D. Dori, *Object-process Methodology: A Holistic Systems Paradigm*. New York, NY, USA: Springer-Verlag, 2002.
[9] N. R. Soderborg, E. F. Crawley, and D. Dori, ''System function and architecture: OPM-based definitions and operational templates,'' *Commun. ACM*, vol. 46, no. 10, pp. 67–72, 2003.
[10] Y. Grobshtein, V. Perelman, E. Safra, and D. Dori, ''Systems modeling languages: OPM versus SysML,'' in *Proc. Int. Conf. Syst. Eng. Modeling*, Mar. 2007, pp. 102–109.
[11] M. Peleg and D. Dori, ''The model multiplicity problem: Experimenting with real-time specification methods,'' *IEEE Trans. Softw. Eng.*, vol. 26, no. 8, pp. 742–759, Aug. 2000.
[12] C. A. R. Hoare, *Communicating Sequential Processes*. Upper Saddle River, NJ, USA: Prentice-Hall, 1985.
[13] R. Milner, *Communication and Concurrency*. Upper Saddle River, NJ, USA: Prentice-Hall, 1989.
[14] R. Milner, *Communicating and Mobile Systems: The Pi-Calculus*. Cambridge, U.K.: Cambridge Univ. Press, 1999.
[15] J. A. Bergstra and J. W. Klop, ''ACP_τ: A universal axiom system for process specification,'' in *Algebraic Methods: Theory, Tools and Applications* (Lecture Notes in Computer Science), vol. 394, M. Wirsing and J. A. Bergstra, Eds. Berlin, Germany: Springer, 1989, pp. 447–463.
[16] N. P. Suh, *The Principles of Design*. London, U.K.: Oxford Univ. Press, 1990.
[17] J. N. Warfield, *An Introduction to Systems Science*. Singapore: World Scientific, 2006.
[18] J. A. Hoffer, J. George, and J. S. Valacich, *Modern Systems Analysis and Design*. London, U.K.: Pearson, 2013.
[19] G. B. Shelly and H. J. Rosenblatt, Eds., *Systems Analysis and Design*. Boston, MA, USA: Course Technology, 2012.
[20] W. R. Beam, *Systems Engineering: Architecture and Design*. New York, NY, USA: McGraw-Hill, 1990.
[21] J. E. Kasser, *A Framework for Understanding Systems Engineering*. North Charleston, SC, USA: BookSurge, 2007.
[22] A. M. Madni and M. Sievers, ''Model-based systems engineering: Motivation, current status, and research opportunities,'' *Syst. Eng.*, vol. 21, no. 3, pp. 172–190, 2018.
[23] R. Cloutier, B. Sauser, M. Bone, and A. Taylor, ''Transitioning systems thinking to model-based systems engineering: Systemigrams to SysML models,'' *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 45, no. 4, pp. 662–674, Apr. 2015.
[24] D. Dori, A. Renick, and N. Wengrowicz, ''When quantitative meets qualitative: Enhancing OPM conceptual systems modeling with MATLAB computational capabilities,'' *Res. Eng. Des.*, vol. 27, no. 2, pp. 141–164, 2016.
[25] 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. 11:1–11:49, 2007.
[26] D. Torre, Y. Labiche, M. Genero, and M. Elaasar, ''A systematic identification of consistency rules for UML diagrams,'' *J. Syst. Softw.*, vol. 144, pp. 121–142, Jun. 2018.
[27] *Object Constraint Language*, Object Manage. Group, Needham, MA, USA, 2014.
[28] *Action Language for Foundational UML*, Object Manage. Group, Needham, MA, USA, 2017.
[29] C.-L. Lazăr, I. Lazăr, B. Pârv, S. Motogna, and I.-G. Czibula, ''Using a fUML action language to construct UML models,'' in *Proc. 11th Int. Symp. Symbolic Numeric Algorithms Sci. Comput.*, Sep. 2009, pp. 93–101.
[30] M. H. Hamilton and W. R. Hackler, ''Universal systems language: Lessons learned from apollo,'' *Computer*, vol. 41, no. 12, pp. 34–43, Dec. 2008.
[31] M. H. Hamilton and W. R. Hackler, ''A formal universal systems semantics for SysML,'' in *Proc. 17th Int. Symp. Int. Council Syst. Eng.*, 2007, pp. 1333–1357.
[32] B. H. Y. Koo, W. L. Simmons, and E. F. Crawley, ''Algebra of Systems: A Metalanguage for Model Synthesis and Evaluation,'' *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 39, no. 3, pp. 501–513, May 2009.
[33] M. Abadi and A. D. Gordon, ''A calculus for cryptographic protocols: The spi calculus,'' in *Proc. 4th ACM Conf. Comput. Commun. Secur.*, 1997, pp. 36–47.
[34] M. Havey, *Essential Business Process Modeling*. Newton, MA, USA: O'Reilly Media, 2009.
[35] F. Ciocchetta and J. Hillston, ''Bio-PEPA: An extension of the process algebra PEPA for biochemical networks,'' *Electron. Notes Theor. Comput. Sci.*, vol. 194, no. 3, pp. 103–117, 2008.
[36] S. L. P. Jones, *The Implementation of Functional Programming Languages*. Upper Saddle River, NJ, USA: Prentice-Hall, 1987.
[37] D. E. Knuth, ''Backus normal form vs. backus naur form,'' *Commun. ACM*, vol. 7, no. 12, pp. 735–736, 1964.
[38] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*. New York, NY, USA: Springer, 2011.

KENG-PEI LIN (M'15) received the B.S. degree in computer science and information engineering from National Chiao Tung University, Hsinchu, Taiwan, in 2005, and the Ph.D. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, in 2011. He is currently an Assistant Professor with the Department of Information Management, National Sun Yat-sen University, Kaohsiung, Taiwan. His research interest includes data mining, machine learning, and system engineering. He is a Member of the IEEE.

WILLIAM S. CHAO received the B.S. degree in communication engineering and the M.S. degree in computer engineering from National Chiao Tung University, Hsinchu, Taiwan, in 1976 and 1981, respectively, and the M.S. and Ph.D. degrees in computer and information sciences from the University of Alabama at Birmingham, AL, USA, in 1985 and 1988, respectively. He is currently an Associate Professor with the Department of Information Management, National Sun Yat-sen University, Kaohsiung, Taiwan. He was with the GE Research and Development Center, NY, USA, as a Computer Scientist, from 1988 to 1990. His research interest includes system architecture and enterprise architecture. He is a member of the Association of Enterprise Architects.